МАГІСТЕРСЬКА РОБОТА
МР.ПМКм-31.00.00.000.ПЗ
Група ПМКм-23-1
Корбеляк Роман
Вікторович
2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інженерної механіки та робототехніки

Кафедра: комп'ютеризованого машинобудування

Корбеляк Роман Вікторович
(прізвище, ім'я, по батькові)

УДК 621:629.3:004.94
(індекс)

МАГІСТЕРСЬКА РОБОТА

Мобільний робот підвищеної прохідності

(назва роботи)

Комп'ютеризовані та роботизовані технології машинобудування
(назва освітньої програми)

131 – Прикладна механіка
(шифр і назва спеціальності)

Р.В. Корбеляк
(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Копей В.Б., професор кафедри КМВ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту
Завідувач кафедри
професор _____ Панчук В. Г.
(посада) (підпис) (дата) (ініціали та прізвище)

Рецензент

_____
(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

м. Івано-Франківськ — 2024 рік

Івано-Франківський національний технічний університет нафти і газу
(повне найменування закладу вищої освіти)
Інститут інженерної механіки та робототехніки
Кафедра комп'ютеризованого машинобудування
Освітній рівень магістр
Спеціальність 131 – Прикладна механіка
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри _____
_____
«____» _____20__ року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТОВІ

Корбеляку Роману Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: Мобільний робот підвищеної прохідності

керівник роботи Копей В.Б., професор кафедри КМВ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від 22.11.2024р. № 780/7

2.Строки подання студентом роботи 15 грудня 2024 р.

3. Вихідні дані до роботи: Інтернет-джерела з описом конструкції мобільних роботів, офіційна документація з симулятара Coppeliasim та Python-пакетів PyODE та SciPy.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд конструкцій мобільних роботів. 2. Симуляція в CoppeliaSim. 3. Програма на основі pyODE для оптимізації конструкції колісного робота. 4. Проектування мобільного робота.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

1. Складальне креслення робота – 1 лист А1. 2. Складальне креслення сферичного колеса – 1 лист А1. 3. Симуляція в Coppeliasim – 1 лист А1. 4. Програма для оптимізації конструкції – 1 лист А1. 5. Принципова електрична схема і керуюча програма – 1 лист А1.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Копей В.Б., професор кафедри КМВ	01.03.24	01.03.24
2	Копей В.Б., професор кафедри КМВ	30.04.24	30.04.24
3	Копей В.Б., професор кафедри КМВ	04.06.24	04.06.24
4	Копей В.Б., професор кафедри КМВ	06.08.24	06.08.24

7. Дата видачі завдання 01.03.24

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Термін виконання етапів роботи	Примітки
1	Огляд конструкцій мобільних роботів	01.04.2024	
2	Симуляція в CoppeliaSim	01.06.2024	
3	Програма на основі ruODE для оптимізації конструкції колісного робота	05.08.2024	
4	Проектування мобільного робота	04.11.2024	
5	Захист магістерської роботи	20.12.2024	

Студент _____ Корбеляк Р.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Копей В.Б.
(підпис) (прізвище та ініціали)

Реферат

Магістерська кваліфікаційна робота на тему: Мобільний робот підвищеної прохідності. Дана робота складається зі 67 аркушів. До неї входять 25 рисунки, 1 таблиця, 2 додатки. Було використано 28 бібліографічних найменувань.

Проаналізовано конструкції платформ мобільних роботів з точки зору їхньої прохідності. Розглянуті переваги та недоліки колісної, гусеничної та крокуючої платформ, а також їхніх підвидів. Виявлено, що платформи зі сферичними колесами є переспективою, але потребує оптимізації багатьох параметрів для забезпечення максимальної прохідності. За допомогою симулятора CoppeliaSim розроблено модель мобільного робота для дослідження його прохідності. За допомогою Pythom-пакетів pyODE і SciPy розроблено програму для симуляції подолання мобільним роботом зі сферичними колесами перешкод та оптимізації конструкції робота на основі алгоритму диференціальної еволюції. Передбачено можливість розпаралелювання аглоритму. За результатами цієї оптимізації спроектовано конструкцію робота з використанням засобів 3D-проектування. Розроблено керуючу програму для мікроконтролера ESP32 та виконано симуляцію керування колесами за допомогою Wokwi.com.

Ключові слова: моделювання, оптимізація, мобільний робот, Python, CoppeliaSim, PyODE.

Студент Корбеляк Р.В.

Summary

Master's qualification work on the topic: Mobile robot with increased cross-country ability. This work consists of 67 pages. It includes 25 figures, 1 table, 2 appendices. 28 bibliographic references were used.

The work analyses the design of mobile robot platforms in terms of their cross-country ability. The advantages and disadvantages of wheeled, tracked and walking platforms, as well as their subtypes, are considered. It is found that platforms with spherical wheels are promising, but require optimisation of many parameters to ensure maximum cross-country ability. A model of a mobile robot was developed using the CoppeliaSim simulator to study its cross-country ability. Using the Pythom packages pyODE and SciPy, a program was developed to simulate the overcoming of obstacles by a mobile robot with spherical wheels and optimise the robot's design based on the differential evolution algorithm. The program provides for the possibility of parallelising the algorythm. Based on the results of this optimisation, the robot design was developed using 3D design tools. The control programme for the ESP32 microcontroller was developed and the wheel control was simulated using Wokwi.com

Keywords: modelling, optimisation, mobile robot, Python, CoppeliaSim, PyODE.

Student Korbeliak R.V.

ЗМІСТ

ВСТУП

Сучасне машинобудування розвивається надзвичайно швидкими темпами, інтегруючи в себе новітні розробки у галузях робототехніки та автоматизації. Серед ключових напрямів — створення промислових і мобільних роботів, які знаходять застосування не лише у виробництві, але й у науці, побуті та логістиці. В епоху науково-технічної революції саме роботи беруть на себе виконання завдань, які часто є трудомісткими, небезпечними або недоступними для людей. Ці завдання охоплюють завантаження й розвантаження обладнання, виконання технологічних операцій (зварювання, фарбування, складання) та роботу у складних або екстремальних умовах, таких як підводне середовище, радіаційні зони, космос чи токсичні середовища.

Один із найперспективніших класів роботів — мобільні платформи. Від стаціонарних роботів вони відрізняються компактними розмірами, здатністю переміщуватися та універсальністю у використанні. Для їхньої ефективної роботи необхідні не лише механічно надійна конструкція та енергоефективні компоненти, але й точні системи управління, що здатні аналізувати оточення в реальному часі. Наприклад, сучасні омніколісні платформи дозволяють значно підвищити маневреність завдяки багатонаправленим колесам, відкриваючи нові можливості для застосування у виробничих і логістичних процесах.

Використання еволюційних алгоритмів для оптимізації конструкції мобільного робота стає інноваційним підходом. Такі алгоритми, завдяки здатності імітувати природні процеси розвитку та адаптації, дозволяють створювати оптимальні конструкції, враховуючи численні параметри та обмеження. Актуальність цього методу підкріплюється прикладами зі світової практики, котрі були проаналізовані у даній роботі.

Метою роботи є розробка оптимальної конструкції мобільного робота, яка поєднує маневреність, енергоефективність і точність. Для досягнення мети поставлені такі завдання: аналіз перспективних конструкцій і технологій у сфері

мобільної робототехніки; вивчення методів оптимізації, включаючи еволюційні алгоритми, та обґрунтування їхнього використання; створення імітаційної моделі для пошуку оптимальних параметрів конструкції; розробка повноцінного дизайну робота, включаючи механічну конструкцію, електричну схему та програму управління; проведення тестування для оцінки результатів оптимізації. Ця робота спрямована на вдосконалення сучасних робототехнічних рішень, підвищення їхньої продуктивності та розширення можливостей застосування в різних сферах.

1 ОГЛЯД КОНСТРУКЦІЙ МОБІЛЬНИХ РОБОТІВ І ЗАСОБІВ СИМУЛЯЦІЇ РУХУ

1.1 Колісна платформа

Багато мобільних роботів, особливо ті, що призначені для переміщення по плоскій поверхні, мають колісну конструкцію. Вони можуть мати два, три, чотири або навіть більше коліс, які дозволяють їм рухатися вперед, назад, вліво та вправо. Така конструкція дозволяє роботам швидко переміщатися та маневрувати [1].

Колісна платформа є одним з найпоширеніших типів конструкцій для мобільних роботів. Вона використовує колеса для руху та маневрування роботом. Розглянемо детальніше складові колісної конструкції :

- Типи коліс: колісні платформи можуть мати різні типи коліс в залежності від вимог і призначення робота. Існують звичайні колеса з пневматичними або гумовими шинами, які забезпечують гладкий рух по рівній поверхні. Також використовуються спеціалізовані колеса, такі як гусеничні колеса для кращої прохідності або м'які колеса для амортизації ударів [1].

- Кількість коліс: мобільні роботи з колісною платформою можуть мати різну кількість коліс. Найпоширеніші варіанти - двоколісні і чотириколісні платформи, але також можуть використовуватись більш складні конфігурації з більшою кількістю коліс для забезпечення стабільності та маневреності.

- Рушійна система: для приведення коліс у рух використовуються різні рушійні системи. Найпоширеніші варіанти - електричні мотори або гідравлічні приводи. Електричні мотори зазвичай живляться від акумуляторів і забезпечують електричну енергію для руху коліс. Гідравлічні приводи використовують рідину під високим тиском для переміщення коліс.

- Керування: колісні платформи можуть мати різні системи керування. Вони можуть бути керовані за допомогою джойстика, радіокерування або програмного забезпечення, що дозволяє автономне пересування.

Автокари, які застосовуються для транспортування вантажів у складських приміщеннях, є прикладом колісної платформи. Враховуючи обмежений простір

між стелажами в таких приміщеннях, автокари повинні мати компактні розміри та високу маневреність. Зазвичай ці транспортні засоби оснащуються класичною конструкцією з поворотним механізмом для керування колесами, розташованими на задній осі. Це дозволяє досягати малого радіуса повороту [2].

Проте, коли потрібно рухатися безпосередньо вбік, без зміни орієнтації всієї платформи, традиційна чотириколісна база з однією поворотною віссю виявляється недостатньою. Щоб забезпечити повноцінну маневреність, необхідний інноваційний підхід до конструкції коліс і самої платформи. Одним із таких рішень є використання так званих «коліс Ілона», які часто називають шведськими колесами [1].

Шведське колесо відрізняється наявністю рівномірно розташованих роликів по ободу, які взаємодіють із поверхнею. Завдяки цій конструкції транспортний засіб, керуючи кожним колесом незалежно, може пересуватися в будь-якому напрямку. Це робить платформу більш універсальною та придатною для складських приміщень із мінімальним простором для маневрів. Сучасні автокари працюють на електричній тязі, що оптимально для експлуатації в закритих приміщеннях, і здебільшого оснащені повним приводом. Інтеграція шведських коліс у конструкцію таких платформ є перспективною, оскільки дозволяє значно підвищити їхню функціональність. Одна з реалізацій омніколісної платформи показана на рисунку 1.1, і вона демонструє, як цей підхід можна застосувати для вирішення задач із транспортування в складських умовах [2].

Багатонаправлені колеса активно застосовуються в робототехніці, промисловості та логістиці вже протягом багатьох років. Вони стали ключовим елементом все направлених конвеєрних систем, які виробляють провідні компанії, а також є популярними у конструкціях роботів. Завдяки можливості рухатися по прямій лінії між двома точками з одночасною корекцією орієнтації, омніколісні роботи чудово виконують завдання, пов'язані з транспортуванням і точним позиціюванням.

Ці платформи широко застосовуються як мобільні основи для маніпуляторів або орієнтації заготовок у виробництві для їх подальшої обробки. Задля підвищення функціональності, конструкції омніколіс можуть оснащуватися системами GPS-навігації та гіроскопами. Це дозволяє досягати більшої точності руху й можливості дистанційного управління.

Важливим елементом такої платформи є система контролю оточення. Подібні системи можуть виявляти перешкоди, зокрема людей, та запобігати зіткненням. Удосконалення, які включають використання систем виявлення й навігації, значно розширюють сферу застосування омніколісних платформ, роблячи їх універсальними в будь-якому виробництві. В результаті це сприяє збільшенню ефективності транспортування вантажів і підвищенню загальної продуктивності підприємств [2].

Рисунок 1.1 - Загальний вигляд омніколісної платформи [3].

1.2 Гусенична платформа

Деякі мобільні роботи, які призначені для руху по нерівним або складним поверхнях, можуть мати гусеничну конструкцію. Гусениці забезпечують кращу стабільність та зчеплення з поверхнею, дозволяючи роботу пройти через перешкоди, такі як нерівності, камені або піски. Мобільні роботи на гусеничній платформі використовуються в різних галузях і мають широкий спектр застосувань. Гусенична платформа надає переваги у теренних умовах, де колісні транспортні засоби можуть бути обмежені.

Приклади застосування мобільних роботів на гусеничній платформі:

- Експлуатація у важкодоступних місцях: Гусеничні роботи можуть використовуватися для досягнення місць, куди важко або неможливо дістатися колісними транспортними засобами. Наприклад, вони також можуть бути використані для різного роду ремонту трубопроводів або ліній електропередач, обстеження складних ландшафтів або проведення рятувальних операцій у важкодоступних місцях.

- Військове застосування: Гусеничні роботи широко використовуються військовими для розвідки, патрулювання, доставки вантажів та бойової підтримки. Вони можуть працювати в різних умовах, включаючи гірські райони, болота або снігові покриви

Компанія Digital Concepts Engineering (DCE) [4] представила публіці свою останню інновацію — безпілотний розвідувальний наземний робот X3 (рисунок 1.2).

Компанія Digital Concepts Engineering (DCE) презентувала свою новітню розробку — безпілотну наземну платформу X3, яка є високомобільним роботом, здатним виконувати численні завдання, зокрема розвідку, спостереження, виявлення цілей і відволікання супротивника. Завдяки гнучкості в конфігурації, платформа може оснащуватися різним обладнанням, включаючи системи для розмінування.

X3 відрізняється потужними технічними характеристиками: можливістю переміщення пересіченою місцевістю з корисним навантаженням до 250 кг і

буксируванням вантажів до 3 тонн. Робот здатний долати складні поверхні, такі як бруд, пісок, каміння, сходи, що робить його ефективним у важких умовах [4] [5].

Рисунок 1.2 – Застосування гусеничної платформи на прикладі безпілотного наземного розвідувального робота X3 [5].

Система управління забезпечує низьку затримку, а режим очікування допомагає економити енергію, що особливо важливо для роботи в віддалених районах. X3 може працювати як у дистанційному режимі, так і з використанням автономних алгоритмів завдяки підтримці роботизованої операційної системи (ROS) [5] [6].

Водночас у межах кластера Brave1 спеціалісти FoxTac розробили транспортер для евакуації поранених із зони бойових дій. Цей дистанційно керований пристрій оснащений повним приводом і адаптивним шасі, що дозволяє перемикатися між колесами й гусеницями. Він здатний пересуватися бездоріжжям, долати важкі умови і має захист від бруду. Радіус дистанційного

керування досягає 700 метрів, а запас ходу — до 10 км, що робить транспортер ефективним і надійним рішенням для критичних ситуацій [6].

- Гірничодобувна промисловість: Гусеничні роботи використовуються у гірничодобувній промисловості для транспортування матеріалів і обладнання, видалення відсічних порід та виконання робіт у складних умовах шахт або кар'єрів.

- Сільське господарство: У сільському господарстві гусеничні роботи можуть використовуватися для засівання полів, збирання врожаю, обробки землі або доставки сільськогосподарської техніки на важкодоступні ділянки

- Будівництво та ремонт інфраструктури: Гусеничні роботи можуть бути використані для будівництва та ремонту доріг, мостів, залізничних шляхів та інших інженерних споруд. Вони забезпечують стабільність і маневреність у важких умовах будівництва.

Гусенична платформа є одним з типів ходових систем, що використовуються в мобільних роботах. Вона складається з низки з'єднаних гусениць, які прокладені навколо рухомого об'єкта, такого як робот або транспортний засіб. Гусениці зазвичай складаються з металевих пластин, з'єднаних гумовими або металевими ланками.

Гусенична платформа складається з таких основних елементів (рисунок 1.3): основною складовою даного типу платформ є гусениці: Гусениці - це набір металевих чи гумових пластин, з'єднаних гумовими або металевими ланками. Вони утворюють "ланцюг" або "змійку", яка обгортається навколо колісної пари або ролика на рухомому об'єкті. Гусениці забезпечують контакт з поверхнею та передають рух об'єкту.

Колісні пари або ролики, які розташовані вздовж гусениць. Вони допомагають підтримувати та керувати гусеницями, дозволяючи рухатися вперед, назад та повороти. Колісні пари або ролики можуть бути приводними або відновлювальними, залежно від конструкції платформи.

Ходова система гусеничної платформи включає механізм для керування рухом гусениць. Вона може включати гідроприводи, електроприводи або механічні системи, які забезпечують рух та маневреність платформи. Рама є структурою, на якій розташована гусенична платформа та інші компоненти робота. Вона забезпечує жорсткість та міцність всієї конструкції, що дозволяє платформі переносити вантажі та працювати в умовах, вимагають стійкості. Приводний механізм забезпечує рух гусениць. Він може включати двигуни, гідравлічні або електричні системи, що перетворюють енергію в рух гусениць.

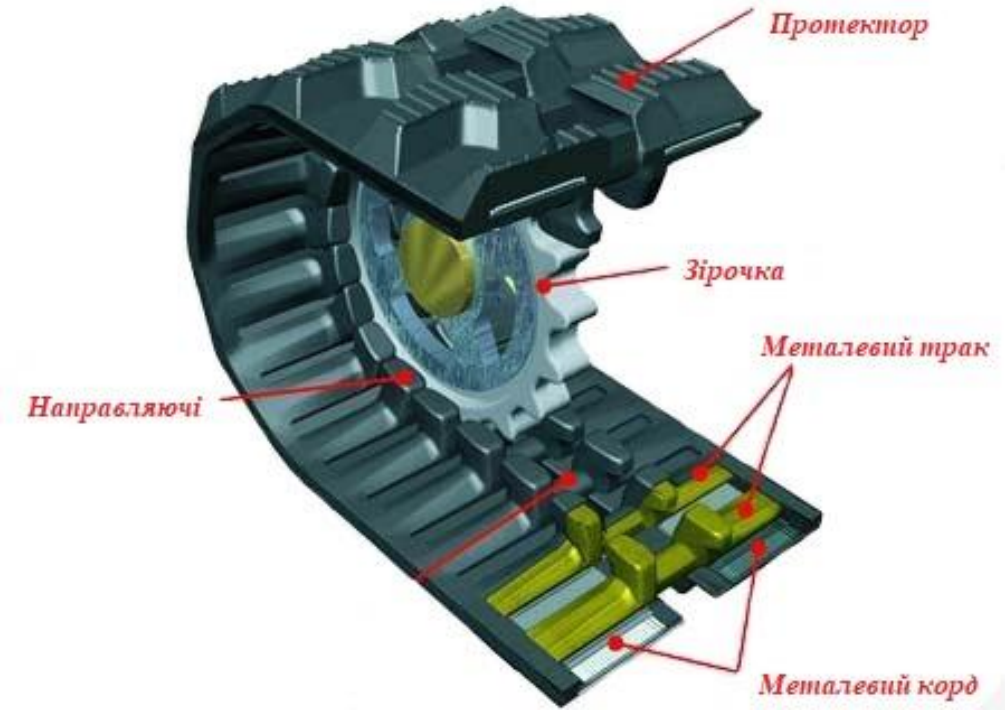


Рисунок 1.3 – Будова гусениці [7].

Основні переваги гусеничної платформи:

- Прохідність: Гусенична платформа забезпечує високу прохідність у різних умовах, включаючи нерівний, м'який або складний рельєф. Гусениці розподіляють вагу робота по більшій площі, допомагаючи уникнути провалів, підскакування або застрягання.

- Стабільність: Гусенична платформа забезпечує високу стабільність робота, особливо на нерівних або крутих поверхнях. Це дозволяє роботу ефективно працювати навіть у ситуаціях, коли колісна платформа може бути нестабільною або нездатною до руху.

- Вантажопідйомність: Гусенична платформа здатна витримувати велику вагу завдяки рівномірному розподілу тиску по грунту. Це робить її ідеальною для перевезення важких вантажів або обладнання.

- Маневреність: Хоча гусенична платформа може бути менш маневрена, ніж колісна, вона все ж здатна повертати на місці або керувати вузькими проходами. Деякі гусеничні роботи також оснащені системами повороту гусениць, що дозволяють їм здійснювати повороти з більшою легкістю.

- Амортизація: Гусенична платформа може забезпечувати кращу амортизацію і знижувати вібрацію при русі по нерівній поверхні. Це допомагає зберегти цілісність робота та зменшити вплив вібрацій на вантаж або обладнання.

Гусенична платформа має свої обмеження, зокрема, повільнішу швидкість руху порівняно з колісною платформою та вищий рівень шуму. Однак, вона є дуже ефективним вибором для роботів, що працюють у важких умовах, де потрібна велика прохідність та стабільність [8] [9].

1.3 Крокуючі роботи

Крокуючі роботи. Особливий розділ робототехніки складають крокуючі системи пересування та засновані на них транспортні машини. Вони є предметом робототехніки тому, що механічні ноги – педипулятори (від латинського слова pes, pedis – нога) – найбільш близькі до іншого основного об'єкту робототехніки – маніпуляторів. Однак значення і потенціал у галузі застосування крокуючих систем машин виходять за межі робототехніки [10].

Спосіб пересування за допомогою ніг (крокування, біг, стрибання), як відомо, є найбільш поширеними в живій природі. Однак у техніці він ще не отримав помітного застосування через складність управління. Такі роботи можуть мати дві або більше ніг і використовувати їх для ходьби, бігу, лазіння або навіть сходження по сходах.

Розвиток робототехніки створив необхідну науково-технічну основу для реалізації цього принципово нового для техніки способу пересування та створення нового типу транспортних машин – крокуючих (рисунок 1.4).

Крокуючий спосіб представляє основний інтерес для руху по заздалегідь непідготовленою місцевістю із перешкодами. Традиційні колісні та гусеничні транспортні машини залишають за собою безперервну колію, витрачаючи на це значно більшу енергію, ніж у разі пересування кроками, коли взаємодія з ґрунтом відбувається лише у місцях упору стопи. Крім цього, крокуючий спосіб пересування має і більшу прохідністю на пересіченій місцевості до можливості пересуватися стрибками, долати перешкоди тощо. При крокуючому способі менше руйнується ґрунт, що, наприклад, важливо у природоохоронних зонах. При рух по досить гладким і підготовленим поверхням цей спосіб поступається колісному в економічності, швидкості пересування та простоті управління [11].

Рисунок 1.4 – Крокуючі роботи компанії Boston Dynamics [12].

Існує багато типів конструкцій для мобільних роботів, включаючи колісні платформи, гусеничні платформи, ноги та дроноподібні роботи. Кожен з цих типів має свої переваги та використання залежно від конкретних умов та завдань.

Проте колісна платформа є однією з найуніверсальніших та широко використовуваних конструкцій. Вона дозволяє роботам швидко переміщатися по плоскій поверхні, маневрувати в різних напрямках і досягати високої швидкості. Колісна платформа також легко налаштовується для різних завдань та може бути використана в різних сферах, включаючи промисловість, медицину, дослідження та багато інших.

Отже, колісна платформа може бути розглянута як універсальна та широко використовувана конструкція для мобільних роботів, що демонструє її популярність і придатність у багатьох сферах діяльності. Саме тому дана платформа була обрана ,як база для реалізації теми бакалаврської роботи та вирішення поставлених ним завдань з оптимізації її конструкції мобільного робота [11] [12].

1.4 Покращення прохідності

Трансформований робот із всенаправленим колесом-ногою

Дослідники з Вустерського політехнічного інституту нещодавно створили OmniWheg, роботизовану систему, яка може адаптувати свою конфігурацію під час навігації в навколишньому середовищі, плавно змінюючи робота з колесами на робота на ногах. Цей робот, представлений у документі IEEE IROS 2022, попередньо опублікованому на arXiv [13], заснований на оновленій версії так званих «whegs», серії механізмів, розроблених для перетворення коліс або крил робота на ноги [14].

«Популярність чотириногих і двоногих роботів зростає, і причиною цього може бути пошук «антропоморфізації», якою зазвичай займається широка аудиторія», — сказав TechXplore професор Андре Розендо, один із дослідників, який розробив робота. «Хоча «здатність їхати всюди, куди ми йдемо» звучить як захоплююче звернення, енергетична вартість ніг дуже висока. Ми, люди, маємо ноги, тому що це те, що нам дала еволюція, але ми б не наважилися створити «легкий автомобіль» оскільки ми знаємо, що ця поїздка не буде такою

комфортною чи енергоефективною, як поїздка на колісному автомобілі» [13] [14].

Ключова ідея нещодавньої роботи Розендо та його колег полягає в тому, що, хоча ноги роблять роботів більш схожими на людину чи тварину, вони не завжди є оптимальним рішенням для того, щоб роботи виконували завдання швидко та ефективно. Замість розробки робота з єдиним механізмом пересування команда таким чином вирішила створити систему, яка може перемикатися між різними механізмами [14].

«Оглядаючи наші будинки та робочі місця, ми бачимо, що наше середовище на 95% рівне, з можливими 5% нерівностей, з якими нам потрібно стикатися під час «переходу» між просторами», — сказав Розендо. «Зважаючи на це, чому б не розробити систему, яка б працювала з «колесною» ефективністю в цих 95% випадків і спеціально переходила до нижчої ефективності в решті 5%?»

Розендо та його колеги вирішили створити колесо, яке могло б змінювати свою конфігурацію, щоб підійматися сходами або обходити інші невеликі перешкоди. Щоб досягти цього, вони дослідили концепцію «whegs» (тобто ноги колеса або ноги-крила), (рисунок 1.5) яка існує вже більше десяти років і з тих пір привернула значну увагу в області робототехніки [13] [14].

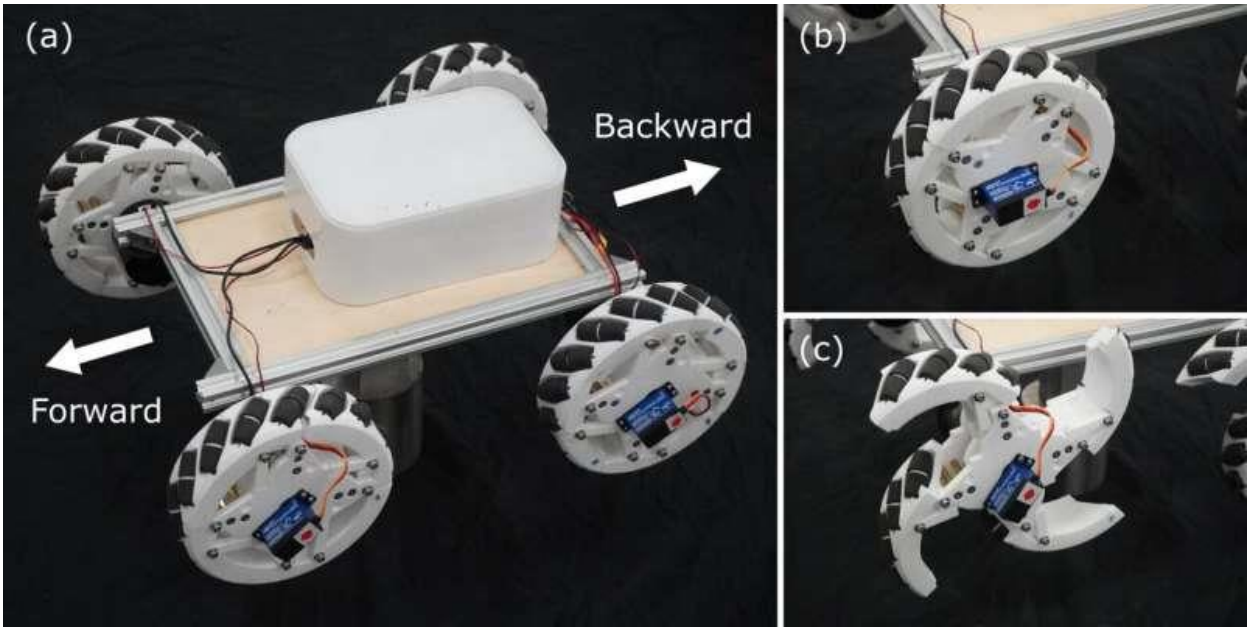


Рисунок 1.5 – Колесо-нога [14].

За останні кілька років було розроблено та випробувано кілька систем колесо-нога. Однак більшість із цих систем не показали себе особливо добре, головним чином через труднощі з координацією правої та лівої сторін системи колесо-нога, які повинні бути ідеально вирівняні, коли робот піднімається сходами.

«Щоб вирішити проблеми з координацією, які зазвичай пов’язані з механізмами «колесо-нога», ми використали всенаправлене колесо», — пояснив Руйсян Цао, провідний студент, який стоїть за створенням. «Це остання частина головоломки, оскільки вона дозволяє роботу вирівнювати на льоту, не обертаючи тіло. Наш робот може рухатися вперед, назад і вбік за дуже низьких витрат енергії, може залишатися в стабільному положенні. без витрат енергії та може швидко підніматися сходами, коли це необхідно».

Для правильної роботи система wheg, створена Розендо та його колегами, потребує додавання одного серводвигуна на колесо та простого алгоритму. Окрім цього, його дизайн простий і простий, тому його можуть легко відтворити інші команди по всьому світу [14] [15].

«Переваг цієї системи настільки багато, а недоліків настільки мало, що ми не можемо не думати, що вони становлять загрозу для «ажіотажу «ногих роботів», який спостерігається в галузі робототехніки», — сказав Цао. «Будь-який додаток-робот, якому в кінцевому підсумку потрібно підніматися сходами, може прийняти цю конструкцію, особливо якщо його поєднати з роботом-маніпулятором, щоб маніпулювати об’єктами під час бігу по рівній землі, зміщуючи свій центр ваги під час підйому сходами» [15].

Дослідники оцінили свою систему OmniWheg у серії експериментів, зосереджених на багатьох реальних сценаріях у приміщенні, таких як обхід перешкод, підйом по сходинках різної висоти та поворот/рух у всіх напрямках. Їхні результати були дуже багатообіцяючими, оскільки їхній робот із колісними ногами міг успішно долати всі типові перешкоди, на яких він тестувався, гнучко

та ефективно адаптуючи свою конфігурацію для ефективного вирішення індивідуальних проблем пересування [15].

У майбутньому систему, створену Розендо та його колегами, можна буде інтегрувати як у існуючих, так і в нових роботів, щоб підвищити їхню ефективність у навігації в приміщенні. Крім того, робота команди може надихнути на розробку подібних систем Wheg на основі всенаправлених коліс.

«У нашій першій ітерації дизайну був використаний досить «дорогий» безщітковий двигун, і тепер ми вважаємо, що легший двигун у поєднанні з редуктором був би ефективнішим», — додав Розендо. «Ми також плануємо додати маніпулятор до основи робота , щоб ми могли перевірити динаміку підйому та спуску сходами з вищим центром ваги» [15].

1.5 Сферичні колеса

У сучасній робототехніці велика увага приділяється розробці нових методів мобільності. Традиційні колісні та гусеничні платформи мають низку обмежень, зокрема низьку маневреність у вузьких просторах і складність виконання точних рухів. Сферичні колеса, як інноваційна технологія, дозволяють значно розширити можливості мобільних роботів завдяки омнінаправленому руху. Розглянемо готові концепти таких коліс та проектів на їхній основі:

Команда Goodyear презентувала концепт сферичних шин Eagle 360 Urban, (рисунок 1.6) які використовують штучний інтелект для вдосконалення функціональності. Основна інновація полягала в здатності шин змінювати малюнок протектора залежно від умов дороги. Наприклад, протектор може ставати опуклим для мокрих доріг або гладким для сухих. Завдяки використанню електроприводів і аналізу дорожніх умов, шини адаптуються, запам’ятовуючи оптимальні налаштування для різних сценаріїв [17].

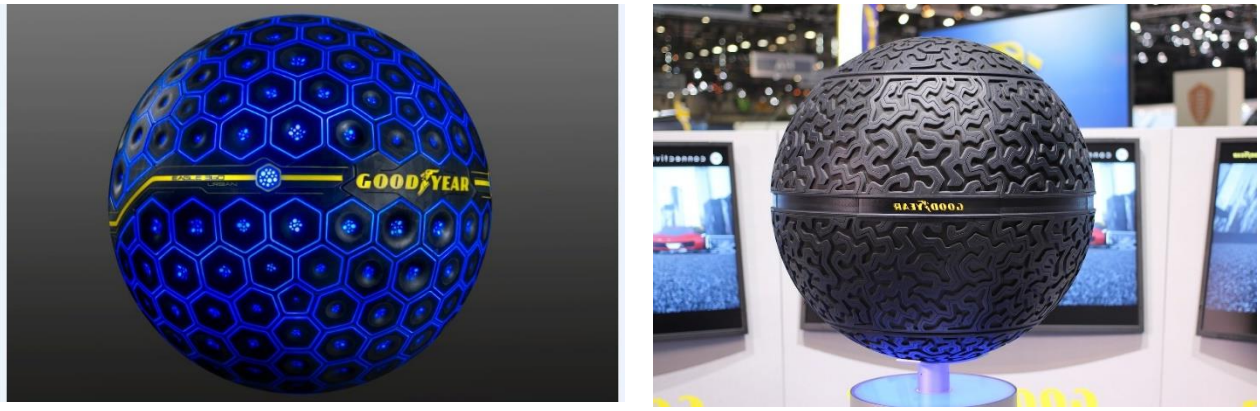


Рисунок 1.6 - Сферична шина Eagle 360 Urban [16] [18].

Окрім цього, шини мають можливість «спілкуватися» через інтернет-з'єднання з іншими такими ж шинами, обмінюючись інформацією про дорожню ситуацію. Це дозволяє транспортним засобам із такими шинами заздалегідь дізнаватися про потенційні небезпеки на дорозі [18].

Ще однією важливою особливістю стала здатність шин виявляти проколи або пошкодження. У разі виникнення дефекту шина повертає сферу так, щоб пошкоджена ділянка не контактувала з дорогою, і автоматично герметизує прокол за допомогою спеціального герметика [16] [17].

Ці нововведення дозволяють досягти високого рівня адаптивності, безпеки та взаємодії між транспортними засобами. Проте, для впровадження цієї технології потрібні спеціальні автомобілі, що стримує її комерційне використання. Незважаючи на потенційно революційний підхід, технологія залишається в концептуальній стадії [17] [18].

Група студентів Інженерного коледжу імені Чарльза В. Девідсона при Університеті штату Сан-Хосе розробляє концептуальний електричний мотоцикл Spherical Drive System (SDS) [19] (рисунок 1.7), який використовує сферичні колеса для всенаправленого руху.

Цей самобалансуючий транспортний засіб покладається на гіроскопічні датчики MEMS, акселерометри та фрикційну систему приводу. Колеса приводяться в рух омніколесами, підключеними до двигунів Animatics, які забезпечують рух у будь-якому напрямку. Для додаткової маневреності

використовується джойстикове керування, що дозволяє рухатися вперед, назад, вбік або розкручувати мотоцикл на місці [19] [20].

Рисунок 1.7 - Концептуальний електричний мотоцикл Spherical Drive System [20]

Особливість конструкції полягає у сферичних колесах, виготовлених із вуглецевого волокна та скловолокна, покритих міцною гумою для покращеного зчеплення. Вони розташовані в маятниках, що дозволяє забезпечити безпечну експлуатацію навіть у разі втрати контакту із землею.

Основні технічні досягнення:

Балансування: Використання акселерометрів та гіроскопів для стабільності за принципом "перевернутого маятника".

Привід: Літій-залізо-магній-фосфатні батареї забезпечують живлення, а мікроконтролери на базі архітектури ARM керують системою.

Підвіска: Використання амортизаторів Fox для плавності ходу.

Поточний стан проєкту: апаратна частина готова на 85%, а програмне забезпечення на 20%.

Цей проєкт демонструє інноваційний підхід до мобільності, пропонуючи унікальні можливості керування та маневреності, але потребує подальших вдосконалень для повноцінної реалізації [20].

Винахідник Деніел Бергет запропонував два варіанти реалізації сферічної системи приводу для автомобіля, що забезпечує рух у різних напрямках.

Перший варіант використання двох пар всенаправлених ведучих коліс і трьох стабілізуючих коліс (рисунок 1.8). Кожне колесо оснащене електричним двигуном, що об'єднано з контролером для точного керування рухом. Усі колеса працюють разом, забезпечуючи багатонаправлений рух транспортної травми, що дозволяє легко маневрувати

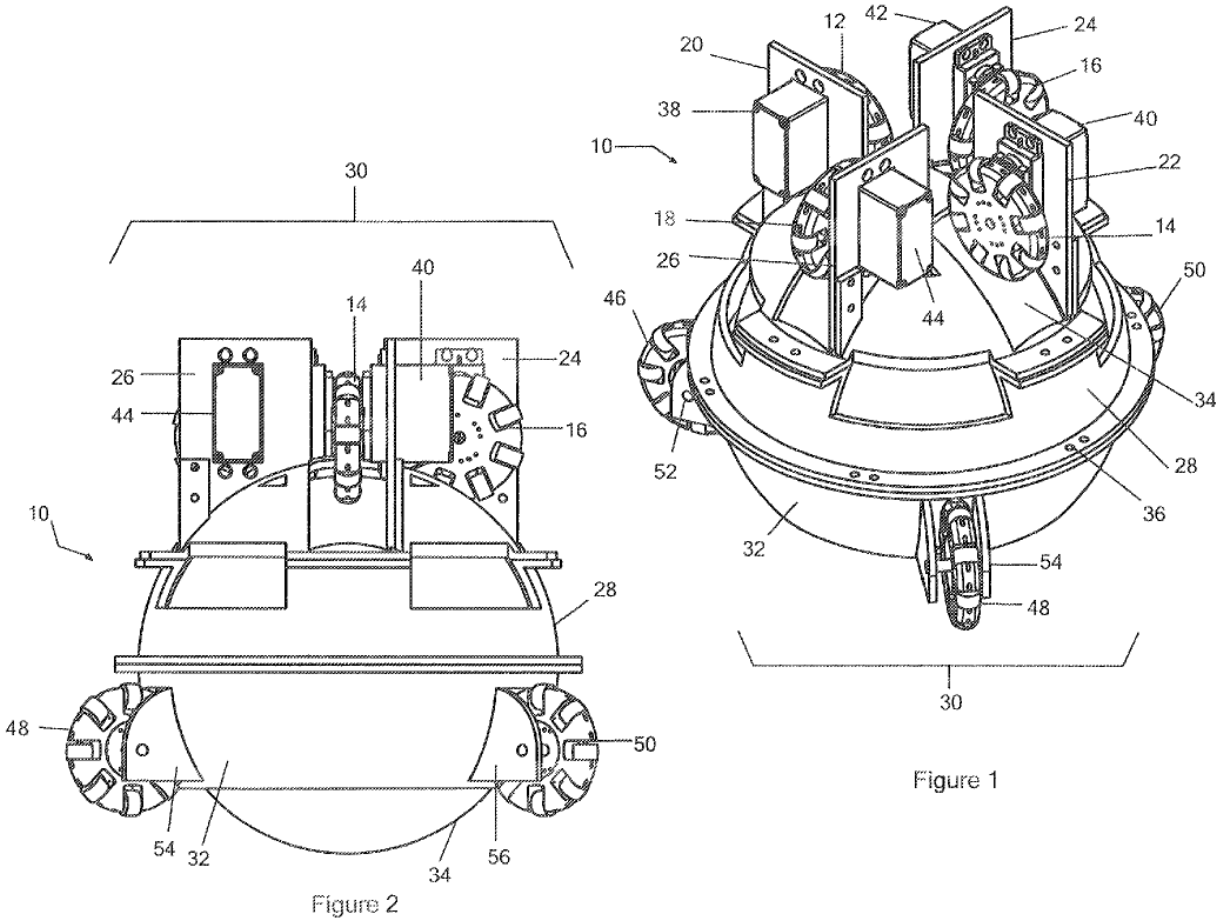


Рисунок 1.8 – Колесо з використанням двох пар всенаправлених ведучих коліс і трьох стабілізуючих коліс [20].

Другий варіант пропонує використання магнітного поля для приводу (рисунок 1.9). У цьому випадку магніти, розташовані в сферічних колах,

взаємодіють з електромагнітами, що знаходяться на зовнішній оболонці. Це дозволяє транспортному ходу рухатися без традиційних коліс, замінюючи їх на електромагнітний привід, що забезпечує маневрування в будь-якому напрямку.

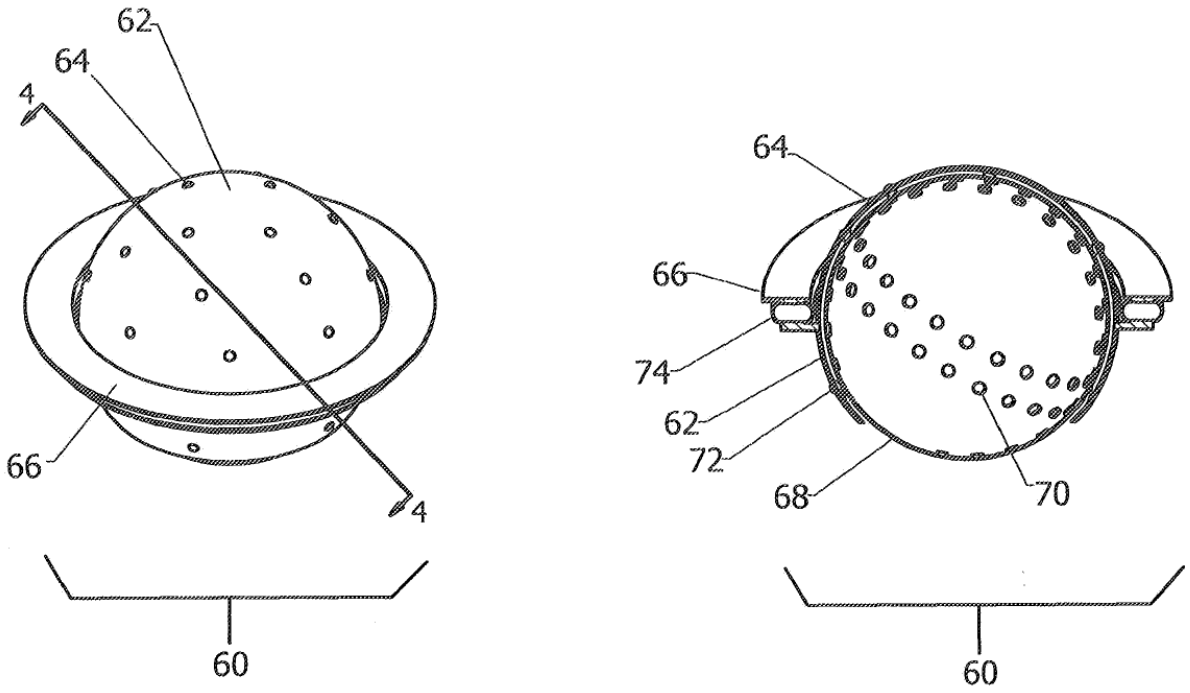


Рисунок 1.9 – Колесо з використанням магнітного поля для приводу [20]

Ці два варіанти можуть значно полегшити маневрування та паркування автомобіля, особливо в обмежених або тісних просторах [20].

1.6 Сферичний робот RT-G

Важко собі представити, що розумні роботи вже зараз приєднуються до поліцейських патрулів і допомагають вистежувати злочинців, Наприклад у Китай офіційно запустили робота під назвою RT-G, який, як ввважається став проривом у правоохоронній сфері. Нижче ми розглянемо його форму, можливості та принцип його роботи. RT-G – це сферичний робот, що нагадує велику кулю, що ззовні виглядає, як сферична шина розміром із звичайну людину. Завдяки своїй певною мірою унікально та динамічною конструкцією він плавно пересувається, як по землі, так і по воді. Його діаметр становить близько 1,5 метра, а вага — кілька сотень кілограмів, що дає йому потрібну стійкість та міцність під час руху. Він євляє собою міцу, ударостійку зовнішню конструкцію

із переднім інтерфейсом, що оснащений передовими камерами та різноманітними датчиками, зокрема зору. Робот був офіційно запущений у 2024 році як частина поліції в місті Ханчжоу, Китай, місті, яке славиться своєю пристрастю до використання сучасних технологій [21].

RT-G оснащений найновішими технологіями штучного інтелекту, такими, як розпізнавання обличчя та аналіз даних, які допомагають відслідковувати злочинців і ідентифікувати розшукуваних осіб. Даний робот може рухатися зі швидкістю до 35 кілометрів на годину та справлятися з суворими умовами. Робот використовує вдосконалену систему навігації, яка може впоратися з перешкодами та різними місцевостями. Він також оснащений інструментами, які можуть перешкоджати правопорушникам, наприклад, видавати звукові чи світлові сигнали, щоб попередити або зупинити злочинців. Крім того, він може надсилати миттєві сповіщення в поліцію при виявленні підозрілої активності та ймовірних злочинців. Ключовою особливістю RT-G є здатність працювати в людних місцях і проводити ретельне спостереження за короткий проміжок часу [21].

З запуском RT-G китайська поліція встановила модель того, чого можна досягти, інтегрувавши технології в безпеку. Робот RT-G, розроблений Logon Technology, був вперше представлений у жовтні 2024 року на Digital Trade World Expo в Ханчжоу, Китай. Цей інноваційний робот здатний ефективно працювати як на землі, так і у воді. Компактна сферична конструкція робота забезпечує самостабілізацію, що робить його ідеальним для різних умов. Він оснащений камерами високої роздільної здатності, GPS і передовими датчиками, які збирають і аналізують дані в реальному часі. На суші він пересувається, зміщуючи центр ваги, а у воді використовує спеціальні гвинти для досягнення швидкості та точності (рисунок 1.10). Робот може розгортати сітку для захоплення злочинців і створений, щоб витримувати значні удари без втрати ефективності. Він здійснює патрулювання громадських місць, таких як вулиці, річки та житлові будинки, допомагає в рятувальних операціях у важкодоступних районах і передає дані безпосередньо в центри управління для аналізу та

використання в операціях безпеки. Цей робот знаменує значний прогрес у використанні технологій для підвищення безпеки та покращення роботи поліції, особливо в середовищах, які вимагають гнучкості та адаптивності [21].

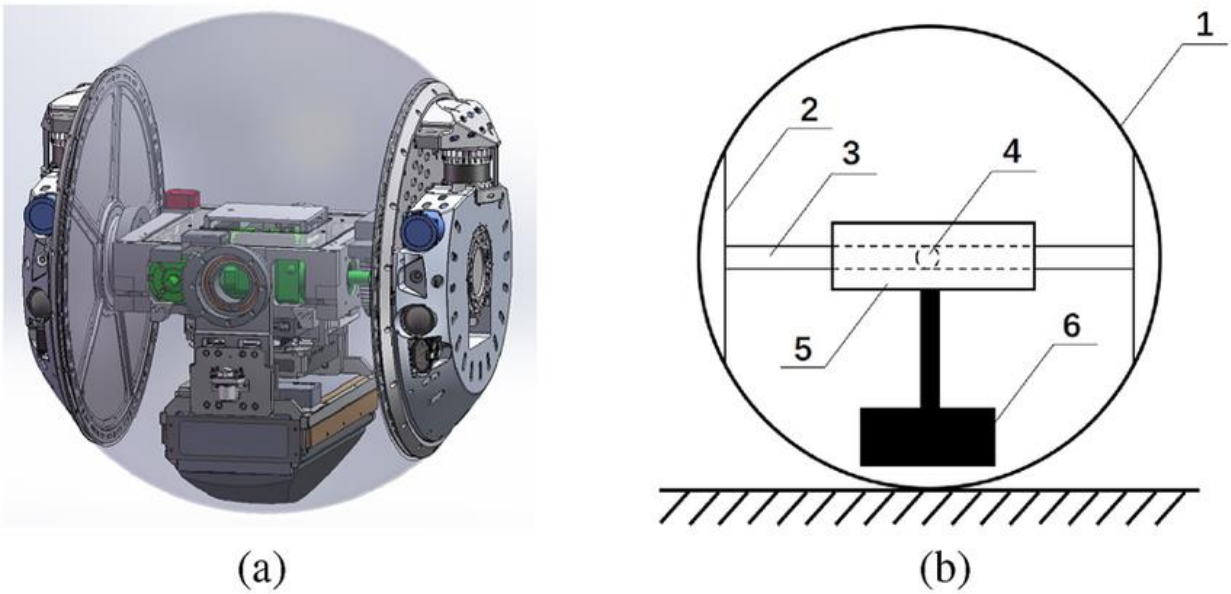


Рисунок 1.10 – Внутрішня будова сферичного робота RT-G [22]

1.7 Огляд симуляторів роботів

Симулятори робототехніки — це програмні інструменти, які використовуються для розробки програмного забезпечення для фізичних роботів, при цьому не вимагаючи наявності реальної машини. Це дозволяє значно заощаджувати кошти та час на етапі розробки. В деяких випадках програми, розроблені в симуляторі, можна без змін перенести на фізичний робот або адаптувати їх для реального використання.

Термін "робототехнічний симулятор" охоплює кілька різних типів програм для моделювання робототехнічних систем. Наприклад, у мобільних роботах симулятори на основі поведінки дозволяють створювати прості світи з твердих об'єктів і джерел світла, а також програмувати роботів для взаємодії з цими об'єктами. Такі симулятори здатні відтворювати більш біотичні дії, на відміну від симуляторів, які орієнтовані на обчислювальні процеси чи бінарні дії. Крім того, поведінкові симулятори можуть вчитися на своїх помилках і проявляти антропоморфні якості, такі як стійкість.

Особливості сучасних симуляторів

Сучасні симулятори зазвичай забезпечують такі можливості:

Швидке створення прототипу робота: дають можливість швидко розробляти робота та його програмне забезпечення.

Використання власних або зовнішніх інструментів для створення роботів.

Фізичні механізми для реалістичних рухів: більшість симуляторів використовують механізми, такі як Bullet, ODE або PhysX, для моделювання фізики руху.

Реалістичний 3D-рендеринг: дозволяє створювати детальні віртуальні середовища для тестування роботів.

Динамічні тіла роботів зі скриптами: симулятори можуть використовувати мови програмування, такі як C, C++, Perl, Python, Java, URBI та MATLAB (наприклад, у Webots), а також C++ в Gazebo.

Симулятори роботів

Віртуальне моделювання робочого середовища та самих роботів відкриває нові можливості для програмування та тестування роботів. Завдяки використанню симуляцій, витрати на розробку зменшуються, а також можна програмувати роботів у автономному режимі, що допомагає уникати часу простою на реальному виробництві. Крім того, віртуальні симулятори дозволяють візуалізувати дії робота ще до виготовлення прототипів, що дає змогу краще зрозуміти й оптимізувати робочі процеси.

Процес програмування для симулятора також є менш складним порівняно з програмуванням для фізичного робота, оскільки віртуальні моделі не піддаються фізичним обмеженням реальних систем. Хоча технології віртуальних симуляцій для робототехніки продовжують розвиватися, багато з них перебувають на етапі вдосконалення, і багато додатків знаходяться лише на початкових етапах розвитку.

Огляд інструментів моделювання робототехніки: основні критерії та рекомендації

Вибір найкращого інструменту для моделювання робототехніки залежить від багатьох факторів, включаючи специфіку завдань, галузь застосування та рівень підтримки симулятора. Як відзначають Джек Коллінз, Шелвін Чанд, Ентоні Вандеркоп та Девід Говард у своїй статті «Огляд фізичних симуляторів для робототехнічних програм» (опублікованій у IEEE Access, Том 9), моделювання є ключовим аспектом робототехніки завдяки його економічності, швидкості та надійності порівняно з використанням фізичних роботів [23].

Автори проводять комплексний аналіз симуляторів, зосереджуючи увагу на таких основних піддоменах робототехніки:

- Наземна мобільна робототехніка;
- Маніпуляційні системи;
- Медична робототехніка;
- Морські роботи;
- Аерокосмічна робототехніка;
- М'яка робототехніка;

Інструменти для навчання та тестування в робототехніці.

Важливими критеріями вибору симулятора є його функціональність, переваги в конкретному піддомені, актуальність для поточних дослідницьких завдань, а також активність підтримки програмного забезпечення. Автори також зазначають, що аналіз історичних і сучасних змагань із робототехніки може бути додатковим джерелом інформації для вибору інструмента [23].

Ключовою особливістю статті є порівняльні таблиці, які структурують рекомендації за субдоменами. Для користувачів, які планують застосовувати ці рекомендації, важливо ознайомитися з деталями обґрунтування рейтингу симуляторів, наведеними в статті. Вона також вказує на прогалини у функціональності симуляторів, що може надати цінну інформацію для подальших досліджень.

Цей огляд допомагає вибрати оптимальний симулятор залежно від специфіки проєкту, забезпечуючи більш глибоке розуміння переваг та обмежень сучасних інструментів моделювання [24].

Simulator	RGBD + LiDAR	Force Sensor	Linear + Cable Acutator	Multi-Body Import	Soft-Body Contacts	DEM Simulation	Fluid Mechanics	Headless Mode	ROS Support	HITL	Teleoperation	Realistic Rendering	Inverse Kinematics
Airsim	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓, unreal	✗
CARLA	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗	✓	✓, unreal	✗
CoppeliaSim	✓	✓	Linear only	✓	✗	✓	✗	✓	✓	✓	✓	✗	✓
Gazebo	✓	✓	Linear only	✓	✗	Through Fluidix	Through Fluidix	✓	✓	✓	✓	✗	✓
MuJoCo	✓	✓	✓	✓	✓	✓	Limited	✓	✗	HAPTIX only	HAPTIX only	✗	✗
PyBullet	✓	✓	Linear only	✓	✓	✓	✗	✓	✗	✗	✓	✗	✓
SOFA	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓, Unity	✗
UWSim	RGBD only	✓	✗	✓	✗	✗	✗	✓	✓	✓	✓	✓, custom	✗
Chrono	✓	✓	✓	✗	✓	✓	✓	✓	✗	✗	✓	✓, offline	✓
Webots	✓	✓	linear	✓	✗	✗	Limited	✓	✓	✗	✓	✗	✗

Рисунок 1.11 - Порівняння функцій між популярними симуляторами для мобільної наземної робототехніки [24].

Simulator	GPS	Tracks	Wheels	Legs	Mecanum / Omni Wheels	Heightmap Import	OpenDrive / OpenStreetMap	Pathplanning	ROS Support	RGBD	LiDAR	Realistic Rendering
Gazebo	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗
CoppeliaSim	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗
Raisim	✗	✗	✓	✓	✗	✓	✗	✗	✗	✓	✓	✓, Unity
Webots	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
PyBullet	✗	✗	✓	✓	✗	✓	✗	✓	✗	✓	✓	✗
CARLA	✓	✗	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓, Unreal
Project Chrono	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓, POV-Ray

Рисунок 1.12 - Порівняння функцій популярних робототехнічних симуляторів, які використовуються для маніпуляцій [24].

Simulator	Random External Forces	RGBD + LiDAR	Force Sensor	Multiple Physics Engines	Realistic Rendering
Raisim	✓	✓	✗	✗	✓, Unity
Gazebo	✓	✓	✓	✓	✗
Nvidia Isaac	✓	✓	✗	✗	✓, Unity + Unreal
MuJoCo	✓	✓	✓	✗	✗
PyBullet	✓	✓	✓	✗	✗
CARLA	✗	✓	✗	✗	✓, Unreal
Webots	✓	✓	✓	✗	✗
CoppeliaSim	✓	✓	✓	✓	✗

Рисунок 1.13 - Порівняння функцій популярних симуляторів, які використовуються в навчанні для робототехніки [24].

CoppeliaSim (раніше відомий як V-REP) [25] — це потужний і універсальний симулятор робототехніки, що забезпечує можливості для моделювання, симуляції та програмування роботів. Завдяки своїм широким функціональним можливостям, він використовується для створення роботизованих систем різного рівня складності. Симулятор підтримує як прості мобільні платформи, так і складні маніпулятори, що дозволяє розробляти та тестувати програми для різних типів роботів, включаючи промислові, автономні, а також роботи, що виконують складні маніпуляції [25].

CoppeliaSim підтримує інтеграцію з кількома мовами програмування, серед яких Python, C++, Lua, Java, MATLAB, а також має можливість написання сценаріїв на Lua. Це дозволяє розробникам вибирати найбільш зручні для них інструменти для програмування, а також інтегрувати симулятор з іншими системами та програмами. Важливою особливістю є використання фізичних рушіїв, таких як Bullet, ODE та Vortex, що дозволяє точно моделювати фізичні взаємодії робота з навколишнім середовищем, враховуючи тертя, зіткнення та інші фізичні процеси [25].

Один із ключових аспектів CoppeliaSim — це можливість візуалізації та моделювання роботів і їх середовищ у тривимірному просторі. Користувач може створювати складні сцени з різноманітними об'єктами, поверхнями та перешкодами, що дозволяє тестувати роботу роботів у різних умовах. Симулятор також підтримує роботу з різними датчиками — від лазерних дальномірів до

камер і ультразвукових сенсорів, що робить можливим тестування алгоритмів автономного руху і навігації в умовах реального середовища[25].

CoppeliaSim дозволяє створювати багатозадачні системи, де кілька роботів можуть одночасно виконувати різні операції, що є важливим для розробки алгоритмів для груп роботів. Крім того, він підтримує інтеграцію з системою ROS (Robot Operating System), що дає змогу розробляти складніші програми та взаємодіяти з реальними роботами. Ця інтеграція також полегшує перехід від симуляції до реальних умов, оскільки алгоритми, розроблені у CoppeliaSim, можна тестувати як на віртуальних, так і на фізичних платформах [25].

Важливою перевагою CoppeliaSim є його здатність до розширення. За допомогою плагінів і модулів, користувач може додавати нові функціональні можливості, підтримку нових типів роботів або інтеграцію з іншими програмами. Це дає змогу адаптувати симулятор під конкретні потреби проектів, які можуть включати як навчальні, так і промислові завдання [25].

Використання CoppeliaSim в навчальних закладах та дослідницьких лабораторіях стало дуже популярним завдяки його зручному інтерфейсу, доступності та можливостям для моделювання складних робототехнічних систем. Завдяки інтеграції з ROS і підтримці різних мов програмування, цей симулятор дозволяє розробляти і тестувати роботів без необхідності створювати фізичні прототипи, що значно знижує витрати і скорочує час на розробку.

CoppeliaSim є важливим інструментом для всіх, хто працює в галузі робототехніки, від студентів і дослідників до інженерів, що розробляють промислові роботизовані системи. Його можливості для створення реалістичних симуляцій та підтримка багатьох мов програмування дозволяють користувачам отримати цінний досвід у розробці і тестуванні складних робототехнічних систем [25].

Open Dynamics Engine [26] (ODE) — це високопродуктивна бібліотека з відкритим кодом, створена на мовах програмування C та C++. Вона призначена для моделювання динаміки твердих тіл та виявлення зіткнень між об'єктами в тривимірному просторі. Головні особливості ODE включають підтримку

моделювання фізичної взаємодії з реалістичним врахуванням тертя та з'єднань. Для роботи з нею пропонується зручний і зрозумілий C/C++ API.

Попри незалежність від графічних рушіїв, ODE містить базовий інструмент рендерингу drawstuff, який використовується для демонстрації та тестування. Ця бібліотека поширюється на умовах відкритих ліцензій BSD і LGPL, що робить її доступною для широкого застосування в дослідженнях і розробках.

ODE надає широкий спектр функцій для моделювання об’єктів у транспортних системах, середовищах віртуальної реальності та симуляції віртуальних істот. Її використовують у багатьох популярних комп’ютерних іграх, як-от STALKER та World of Goo, а також у 3D-інструментах моделювання й віртуальних симуляторах. Завдяки стабільності, платформній незалежності та зрілості, ODE активно використовується як у сфері робототехніки, так і у творчих та наукових застосуваннях [26].

Система підтримує різні геометричні форми, такі як сфери, циліндри, паралелепіпеди,капсули, трикутні сітки, карти висот, куби та інші. Це дозволяє використовувати ODE у багатьох програмах і проєктах, де потрібна симуляція динаміки, зокрема в додатках для робототехніки та у ігрових рушіях.

Бібліотека що була розроблена в 2001 році й знайшла своє використання в багатьох проєктах, наприклад, у таких відеоіграх, як Assetto Corsa, BloodRayne 2, STALKER і World of Goo. Також її інтегрують у середовища моделювання, зокрема:

- Player Project,
- Webots,
- OpenSimulator,
- CoppeliaSim.

Особливо популярною ODE є в робототехніці. Її застосовують для симуляцій мобільних роботів, захоплення об'єктів і маніпуляцій.

Незважаючи на свою функціональність, ODE має певні недоліки та ряд обмежень зокрема:

- Спрощений підхід до обчислення тертя.
- Обмежена підтримка демпфування (затухання) в з’єднаннях.

У цілому ODE є універсальним і доступним інструментом для задач фізичного моделювання, однак для конкретних сценаріїв може знадобитися розширення функціоналу або вибір інших програм.

Після аналізу програм для симуляції було сформовано таблицю (таблиця 1) з їхніми особливостями, перевагами та недоліками.

Таблиця 1 – Огляд та аналіз програм для симуляції

Проаналізовані програми	Фізичний рушій	Механізм 3D візуалізації	Динамічне уникнення зіткнень	Підтримка усіх сімей роботів	Підтримка 2D та 3D лазерних сканерів	Зіткнення
Gazebo	ODE , Bullet , Simbody , DART	OGRE	Так	Так	Так	Так
RoboDK	Плагін Gravity	OpenGL	Так	Немає	Так	Так
SimSpark	ODE	внутрішній	Немає	Частково	Немає	Так
Webots	Fork of ODE	Внутрішній (WREN)	Так	Так	Так	Так
OpenRave	ODE, Bullet	Coin3D , OpenSceneGraph	Немає	Так	Так	Так
Coppelia Sim	MuJoCo, Bullet, ODE, Vortex, Newton	внутрішній	Так	Так	Так	Так

2 СИМУЛЯЦІЯ МОБІЛЬНОГО РОБОТА В COPPELIASIM

Додамо примітивну сферу діаметром 0.2 до сцени за допомогою [Add > Primitive shape > Sphere] (Додати > Примітивна форма > Сфера). Налаштуємо елемент X-size на 0.2, а потім натиснемо OK. Створена сфера за замовчуванням з'явиться у шарі видимості 1 і буде динамічною та здатною реагувати (оскільки ми залишили увімкненим пункт Create dynamic and responsable shape (Створити динамічну та здатну реагувати фігуру)). Це означає, що тіло BubbleRob'a падатиме і реагуватиме на зіткнення з іншими фігурами, що реагують (тобто симулюються фізичним рушієм). Ми бачимо, що це діалогове вікно динаміки фігури: пункти Body is respondable і Body is dynamic увімкнено. Запускаємо симуляцію (кнопкою на панелі інструментів або натисканням <контроль-пробіл> у вікні сцени) і копіюємо та вставляємо створену сферу (за допомогою [Edit > Copy object(s)], потім [Edit -> Paste buffer], або за допомогою <контроль-c>, потім <контроль-v>): обидві сфери відреагують на зіткнення і відкотяться убік. Ми зупиняємо симуляцію: продубльовану сферу буде автоматично видалено. Цю поведінку за замовчуванням можна змінити у діалоговому вікні симуляції.

Ми також хочемо, щоб тіло BubbleRob можна було використовувати в інших розрахункових модулях (наприклад, для обчислення відстані). Для цього ми вмикаємо опції Collidable (Зіткнення), Measurable (Вимірювання) та Detectable (Виявлення) у загальному діалоговому вікні властивостей об'єктів сцени для цієї фігури, якщо їх ще не ввімкнено. При бажанні, ми також можемо змінити візуальний вигляд нашої сфери у діалоговому вікні фігури.

Тепер відкриваємо діалогове вікно положення на вкладці перекладу, вибираємо сферу, що представляє тіло BubbleRob'a, і вводимо 0.02 для параметра Along Z. Переконаємося, що для пункту Relative to (Відносно до) встановлено значення World (Світ). Потім натискаємо кнопку Translate selection (Перекласти виділення). Це переведе всі виділені об'єкти на 2 см вздовж абсолютної осі Z, і ефективно трохи підніме нашу сферу. В ієрархії сцени ми двічі клацаємо

псевдонім сфери, щоб відредагувати його. Вводимо bubbleRob і натискаємо Enter.

Далі ми додамо датчик наближення, щоб BubbleRob знав, коли він наближається до перешкод: обираємо [Add > Proximity sensor > Cone type] (Додати > Датчик наближення > Конус). У діалоговому вікні орієнтації на вкладці Орієнтація вводимо 90 для параметра Навколо Y і для параметра Навколо Z, потім натискаємо кнопку Повернути виділення. У діалоговому вікні положення на вкладці положення вводимо 0.1 для координати X і 0.12 для координати Z. Тепер датчик наближення правильно розташовано відносно тіла BubbleRob. Двічі клацніть на іконці датчика наближення в ієрархії сцени, щоб відкрити діалогове вікно його властивостей. Натискаємо кнопку Показати параметр гучності, щоб відкрити діалогове вікно гучності датчика наближення. Встановлюємо значення параметрів Зсув на 0.005, Кут на 30 і Діапазон на 0.15. Потім у діалоговому вікні датчика наближення натискаємо Показати параметри виявлення. Відкриється діалогове вікно параметрів виявлення датчика наближення. Знімаємо позначку з пункту Не дозволяти виявлення, якщо відстань менша, а потім знову закриваємо це діалогове вікно. В ієрархії сцени двічі клацніть псевдонім датчика наближення, щоб відредагувати його. Вводимо sensingNose і натискаємо клавішу Enter.

Виділяємо sensingNose, потім виділяємо bubbleRob, а потім натискаємо [Edit > Set parent, keep pose(s)]. Це прикріпить сенсор до тіла робота. Ми також могли б перетягнути sensingNose на bubbleRob в ієрархії сцени. Це те, що ми маємо зараз:

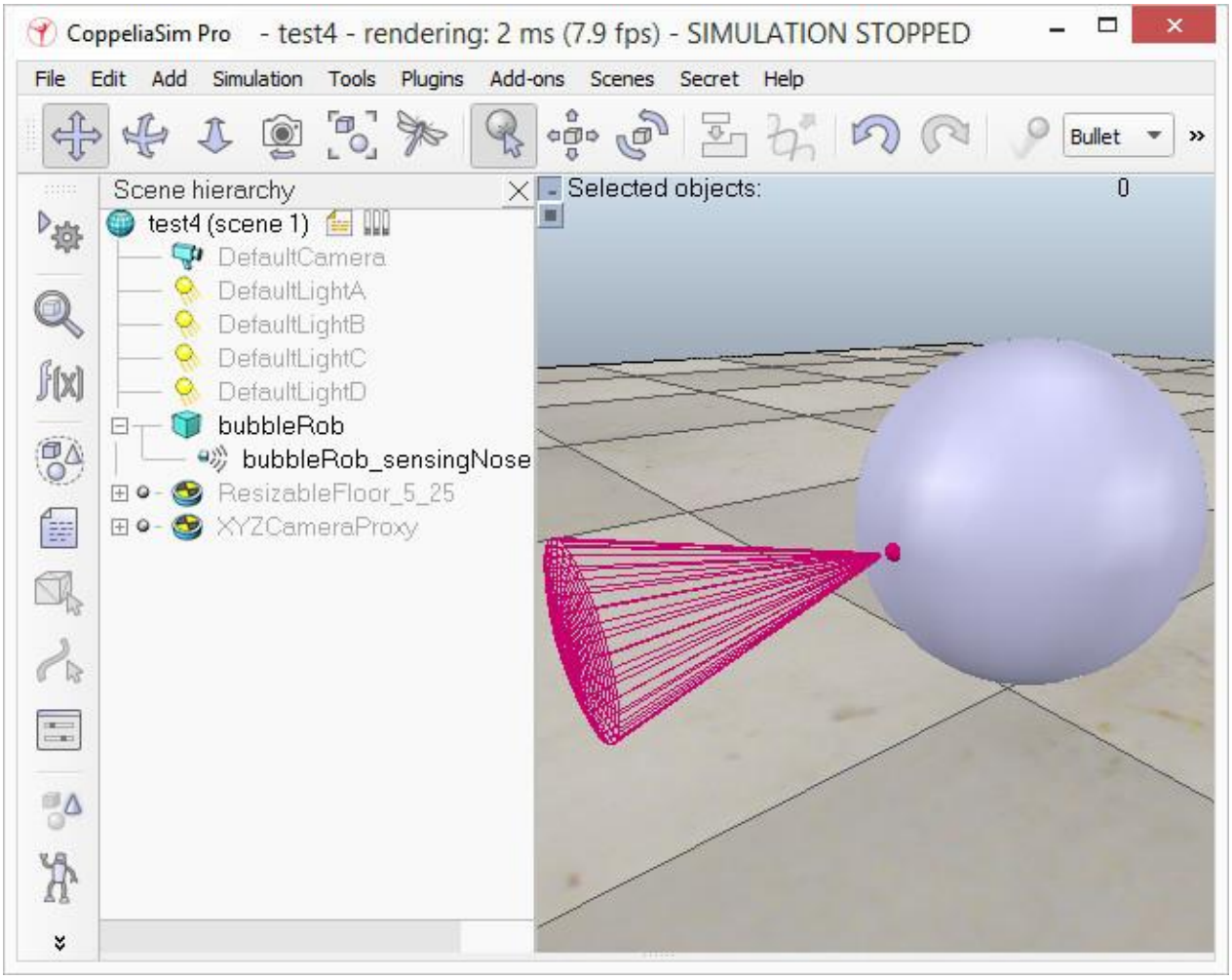


Рисунок 2.1 - Датчик наближення прикріплено до тіла bubbleRob

Далі ми подбаємо про колеса BubbleRob'a. Створюємо нову сцену за допомогою [File > New scene]. Часто буває дуже зручно працювати з декількома сценами, щоб візуалізувати і працювати тільки з певними елементами. Додаємо примітивний циліндр з розмірами (0.08,0.08,0.02). Для тіла BubbleRob'a ми вмикаємо Collidable, Measurable та Detectable у загальному діалозі властивостей об'єктів сцени для цього циліндра, якщо вони ще не ввімкнені. Потім встановлюємо абсолютне положення циліндра (0.05,0.1,0.04) і його абсолютну орієнтацію (-90,0,0). Змінюємо псевдонім на leftWheel. Копіюємо і вставляємо колесо і встановлюємо абсолютну координату Y копії -0.1. Перейменовуємо копію на rightWheel. Виділяємо два колеса, копіюємо їх, повертаємося до сцени 1 і вставляємо колеса.

Тепер нам потрібно додати шарніри (або двигуни) для коліс. Натискаємо [Add > Joint > Revolute] (Додати > Шарнір > Обертання), щоб додати обертовий

шарнір до сцени. Здебільшого під час додавання нового об'єкта на сцену об'єкт з'являється на початку світу. Тримаємо шарнір виділеним і натискаємо ліве коліщатко миші. У діалоговому вікні положення на вкладці Position (Положення) натискаємо кнопку Apply to selection (Застосувати до виділеного): це позиціонує з'єднання в центрі лівого колеса. Потім у діалоговому вікні «Орієнтація» на вкладці «Орієнтація» робимо те саме: з'єднання орієнтується так само, як і ліве колесо. Перейменовуємо з'єднання на leftMotor. Тепер двічі клацніть піктограму зчленування в ієрархії сцени, щоб відкрити діалогове вікно зчленування. Потім натискаємо кнопку Показати динамічні параметри, щоб відкрити діалогове вікно динаміки зчленування. Вибираємо режим керування швидкістю. Повторюємо ту саму процедуру для правого двигуна і перейменовуємо його на rightMotor. Тепер приєднуємо ліве колесо до лівого мотора, праве колесо до правого мотора, а потім приєднуємо обидва мотори до bubbleRob. Ось що у нас вийшло:

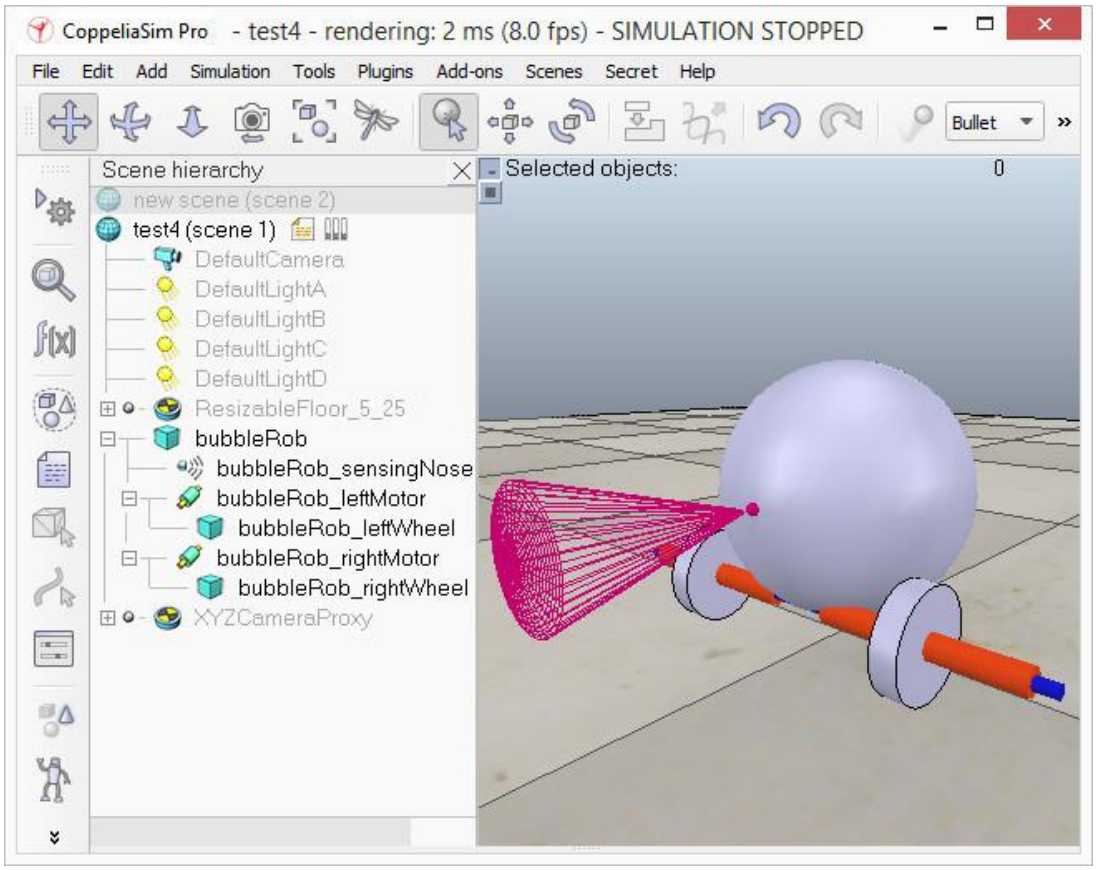


Рисунок 2.2 - Датчик наближення, двигуни та колеса

Запускаємо симуляцію і помічаємо, що робот падає назад. Нам все ще не вистачає третьої точки контакту з підлогою. Тепер ми додаємо невеликий повзунок (або кастер). У новій сцені ми додаємо примітивну сферу діаметром 0,05 і робимо сферу зіштовхуваною, вимірюваною і виявленою (якщо це ще не ввімкнено), а потім перейменовуємо її на слайдер. У діалоговому вікні динаміки форми встановлюємо параметр Material (Матеріал) на noFrictionMaterial (Без тертя). Щоб жорстко зв'язати повзунок з рештою робота, додаємо об'єкт датчика сили за допомогою [Add > Force sensor] (Додати > Датчик сили). Перейменовуємо його на connection і зсуваємо вгору на 0,05. Приєднуємо повзунок до датчика сили, копіюємо обидва об'єкти, повертаємося до сцени 1 і вставляємо їх. Зсуваємо датчик сили на -0.07 вздовж абсолютної осі X і приєднуємо його до тіла робота. Якщо запустити симуляцію зараз, то можна помітити, що повзунок трохи рухається відносно тіла робота: це відбувається тому, що обидва об'єкти (тобто повзунок і bubbleRob) зіштовхуються один з одним. Щоб уникнути дивних ефектів під час симуляції динаміки, ми повинні повідомити CoppeliaSim, що обидва об'єкти не стикаються один з одним, і ми зробимо це наступним чином: у діалоговому вікні динаміки фігури для повзунка ми встановлюємо локальну маску, що реагує, на 00001111, а для bubbleRob встановлюємо локальну маску, що реагує, на 11110000. Якщо ми запустимо симуляцію знову, то побачимо, що обидва об'єкти більше не заважають. Ось що ми маємо зараз:

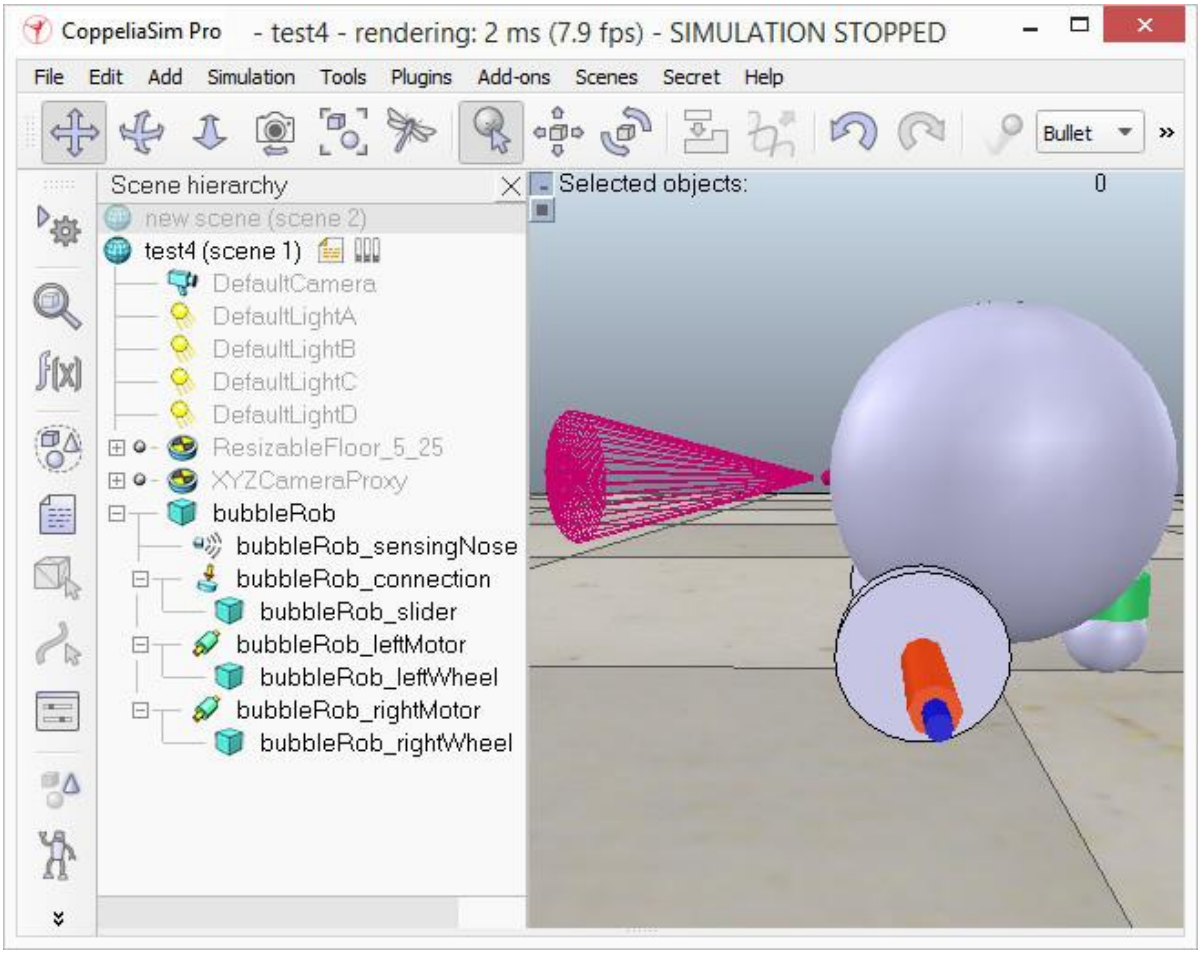


Рисунок 2.3 - Датчик наближення, двигуни, колеса та повзунок.

Запускаємо симуляцію ще раз і помічаємо, що BubbleRob злегка рухається, навіть із заблокованим двигуном. Ми також спробували запустити симуляцію з різними фізичними рушіями: результат буде різним. Стабільність динамічних симуляцій тісно пов'язана з масами та матрицями інерції задіяних нестатичних фігур. Для пояснення цього ефекту уважно прочитайте відповідні розділи. Тепер спробуємо виправити цей небажаний ефект. Виділяємо два колеса та повзунок і збільшуємо їхні маси у 8 разів за допомогою [Edit > Shape mass and inertia > scale mass...] (Редагування > Маса та інерція фігури > Масштабна маса...). Те саме робимо з матрицями інерції 3 вибраних фігур ([Edit > Shape mass and inertia > scale inertia...]), після чого знову запускаємо симуляцію: стабільність покращилася. У діалоговому вікні спільної динаміки встановлюємо цільову швидкість 50 для обох двигунів. Запускаємо симуляцію: BubbleRob рухається

вперед і врешті-решт падає з підлоги. Ми скидаємо значення параметра Target velocity (Цільова швидкість) на нуль для обох двигунів.

Об'єкт bubbleRob лежить в основі всіх об'єктів, які пізніше сформують модель BubbleRob. Ми визначимо модель трохи пізніше. Далі ми додамо об'єкт-графік до BubbleRob, щоб відобразити відстань до нього. Натискаємо [Add > Graph] і перейменовуємо його на graph. Прикріплюємо графік до bubbleRob і встановлюємо абсолютні координати графіка (0,0,0.005).

Тепер ми встановлюємо цільову швидкість одного з двигунів на 50, запускаємо симуляцію і бачимо траєкторію руху BubbleRob'a, яка відображається в сцені. Потім ми зупиняємо симуляцію і скидаємо цільову швидкість двигуна на нуль.

Додамо примітивний циліндр з такими розмірами: (0.1, 0.1, 0.2). Ми хочемо, щоб цей циліндр був статичним (тобто не зазнавав впливу гравітації або зіткнень), але все ж таки реагував на зіткнення з нестатичними фігурами, що реагують на нього. Для цього ми вимикаємо опцію Body is dynamic у діалоговому вікні динаміки фігури. Ми також хочемо, щоб наш циліндр був зіштовхуваним, вимірюваним і виявленим. Це можна зробити у діалоговому вікні властивостей об'єкта сцени. Тепер, поки циліндр все ще виділено, натискаємо кнопку на панелі інструментів переведення об'єктів: 

Тепер ми можемо перетягнути будь-яку точку на сцені: циліндр буде слідувати за рухом, але завжди буде змушений зберігати ту саму координату Z. Ми копіюємо і вставляємо циліндр кілька разів, і переміщуємо їх у позиції навколо BubbleRob'a (найзручніше це робити, дивлячись на сцену зверху). Під час переміщення об'єкта, утримуючи клавішу shift, можна робити менші кроки зсуву. Утримування клавіші ctrl дозволяє переміщатися в напрямку, ортогональному до звичайного напрямку (напрямків). Закінчивши, знову натискаємо кнопку панелі інструментів панорамування камери: 

Встановлюємо цільову швидкість 50 для лівого двигуна і запускаємо симуляцію: тепер на графіку відображається відстань до найближчої перешкоди,

а відрізок відстані видно і в сцені. Ми зупиняємо симуляцію і скидаємо цільову швидкість до нуля.

Тепер нам потрібно завершити BubbleRob як визначення моделі. Вибираємо основу моделі (тобто об'єкт bubbleRob), а потім встановлюємо прапорець «Об'єкт є основою моделі» у діалоговому вікні властивостей об'єктів сцени: тепер з'являється обмежувальна рамка, яка охоплює всі об'єкти в ієрархії моделі. Виділяємо два з'єднання, датчик наближення та графік, вмикаємо пункт Ignored by model bounding box (Ігнорується рамкою моделі) та натискаємо Apply to selection (Застосувати до виділення) у тому ж діалоговому вікні: тепер рамка моделі ігнорує два з'єднання та датчик наближення. У тому ж діалоговому вікні вимикаємо шар видимості камери 2 і вмикаємо шар видимості камери 10 для двох з'єднань і датчика сили: це ефективно приховує два з'єднання і датчик сили, оскільки шари 9-16 вимкнені за замовчуванням. У будь-який момент ми можемо змінити шари видимості для всієї сцени. Щоб завершити визначення моделі, ми виділяємо датчик зору, два колеса, повзунок і графік, а потім вмикаємо пункт Select base of model instead: якщо ми спробуємо виділити об'єкт у нашій моделі в сцені, замість нього буде виділено всю модель, що є зручним способом обробки і маніпулювання всією моделлю як єдиним об'єктом. Крім того, це захищає модель від ненавмисної модифікації. Окремі об'єкти моделі все ще можна виділити у сцені, клацнувши їх з натиснутою клавішею Shift, або просто вибравши їх в ієрархії сцени. Нарешті, ми згортаємо дерево моделі в ієрархії сцени.

Далі ми додамо датчик зору, у тій самій позиції, що й датчик наближення BubbleRob'a. Знову відкриваємо ієрархію моделі, натискаємо [Add > Vision sensor > Perspective type] (Додати > Датчик зору > Тип перспективи), приєднуємо датчик зору до датчика наближення і встановлюємо локальну позицію датчика зору на (0,0,0). Ми також переконуємося, що датчик зору не видно, що він не є частиною рамки моделі, і що при натисканні на нього, замість нього буде вибрано модель. Для того, щоб налаштувати датчик зору, відкриваємо діалог його властивостей. Встановлюємо для параметра Далека площина відсікання

значення 1, а для параметрів Роздільна здатність x і Роздільна здатність y - 256 і 256. Додаємо до сцени плаваючий вид, і над щойно доданим плаваючим видом натискаємо правою кнопкою миші [Спливне меню > Вид > Зв'язати вид з вибраним датчиком зору] (під час цього процесу слід переконатися, що датчик зору вибрано).

Приєднуємо скрипт симуляції до датчика зору, натиснувши [Додати > Скрипт > Скрипт симуляції > Безпотоковий > Lua]. Двічі клацаємо іконку нового сценарію в ієрархії сцен: відкривається щойно доданий сценарій моделювання. Копіюємо та вставляємо наступний код до редактора скриптів, а потім закриваємо його:

```
#python

def sysCall_init():
    sim = require('sim')
    simVision = require('simVision')

def sysCall_vision(inData):

simVision.sensorImgToWorkImg(inData['handle']) #скопіювати зображення
датчика зору в робоче зображення
    simVision.edgeDetectionOnWorkImg(inData['handle'], 0.2) # виконати
визначення країв на робочому зображенні
    simVision.workImgToSensorImg(inData['handle']) # копіювати робоче
зображення в буфер зображення датчика зору
```

Щоб побачити зображення датчика зору, ми починаємо симуляцію, а потім знову зупиняємо її.

Останнє, що нам потрібно для нашої сцени, це сценарій моделювання , який керує поведінкою BubbleRob . Ми вибираємо bubbleRob і натискаємо [Додати > Сценарій > Сценарій симуляції > Безпотоковий > Lua]. Ми двічі

клацаємо піктограму нового сценарію в ієрархії сцени та копіюємо та вставляємо наступний код у редактор сценаріїв , а потім закриваємо його:

```
#python

import math

def sysCall_init():
    # Це виконується рівно один раз, коли цей сценарій виконується вперше
    sim = require('sim')
    simUI = require('simUI')
    self.bubbleRobBase = sim.getObject('..') # Це дескриптор bubbleRob
    self.leftMotor = sim.getObject("../leftMotor") # Ручка лівого двигуна
    self.rightMotor = sim.getObject("../rightMotor") # Ручка правого двигуна
    self.noseSensor = sim.getObject("../sensingNose") # Ручка датчика наближення
    self.minMaxSpeed = [50*math.pi/180, 300*math.pi/180] # Мінімальна та максимальна швидкість для кожного двигуна
    self.backUntilTime = -1 # Повідомляє, чи знаходиться bubbleRob у прямому чи зворотному режимі
    self.robotCollection = sim.createCollection(0)
    sim.addItemToCollection(self.robotCollection, sim.handle_tree, self.bubbleRobBase, 0)
    self.distanceSegment = sim.addDrawingObject(sim.drawing_lines, 4, 0, -1, 1, [0, 1, 0])
    self.robotTrace = sim.addDrawingObject(sim.drawing_linestrip + sim.drawing_cyclic, 2, 0, -1, 200, [1, 1, 0], None, None, [1, 1, 0])
    self.graph = sim.getObject('../graph')
    # sim.destroyGraphCurve(self.graph, -1)
    self.distStream = sim.addGraphStream(self.graph, 'bubbleRob clearance', 'm', 0, [1, 0, 0])
```

```
        # Створення власного інтерфейсу користувача UI:
        xml = '<ui title="' + sim.getObjectAlias(self.bubbleRobBase, 1) + ' speed" closeable="false" resizable="false" activate="false">'
        xml = xml + '<hslider minimum="0" maximum="100" on-change="speedChange_callback" id="1"/>'
        xml = xml + '<label text="" style="* {margin-left: 300px;}"></label></ui>'
        self.ui = simUI.create(xml)
        self.speed = (self.minMaxSpeed[0] + self.minMaxSpeed[1]) * 0.5
        simUI.setSliderValue(self.ui, 1, 100 * (self.speed - self.minMaxSpeed[0]) / (self.minMaxSpeed[1] - self.minMaxSpeed[0]))

    def sysCall_sensing():
        result, distData, *_ = sim.checkDistance(self.robotCollection, sim.handle_all)
        if result > 0:
            sim.addDrawingObjectItem(self.distanceSegment, None)
            sim.addDrawingObjectItem(self.distanceSegment, distData)
            sim.setGraphStreamValue(self.graph,self.distStream, distData[6])
        p = sim.getObjectPosition(self.bubbleRobBase)
        sim.addDrawingObjectItem(self.robotTrace, p)

    def speedChange_callback(ui, id, newVal):
        self.speed = self.minMaxSpeed[0] + (self.minMaxSpeed[1] - self.minMaxSpeed[0]) * newVal / 100

    def sysCall_actuation():
        result, *_ = sim.readProximitySensor(self.noseSensor) # Зчитування датчика наближення
        # Якщо ми щось виявили, ми встановлюємо зворотний режим:
     якщо результат > 0:
            self.backUntilTime = sim.getSimulationTime() + 4
```

```
        if self.backUntilTime < sim.getSimulationTime():
            # У режимі вперед ми просто рухаємося вперед із бажаною швидкістю
            sim.setJointTargetVelocity(self.leftMotor, self.speed)
            sim.setJointTargetVelocity(self.rightMotor, self.speed)
        else:
            # У зворотному режимі ми просто виконуємо резервне копіювання за кривою зі зниженою швидкістю
            sim.setJointTargetVelocity(self.leftMotor, -self.speed / 2)
            sim.setJointTargetVelocity(self.rightMotor, -self.speed / 8)

    def sysCall_cleanup():
        simUI.destroy(self.ui)
```

Ми запускаємо симуляцію. BubbleRob тепер рухається вперед, намагаючись уникати перешкод (дуже просто). Поки симуляція все ще працює, змініть швидкість BubbleRob і скопіюйте/вставте її кілька разів. Майте на увазі, що функція обчислення мінімальної відстані може сильно сповільнювати симуляцію залежно від середовища.

3 ПРОГРАМА НА ОСНОВІ PYODE ДЛЯ ОПТИМІЗАЦІЇ КОНСТРУКЦІЇ КОЛІСНОГО РОБОТА

3.1. Модуль для симуляції руху на основі pyODE

Існує клас мобільних роботів, які повинні ефективно пересуватись по бездоріжжю та поверхнях з різними перешкодами. Відомо, що найкращі характеристики прохідності мають гусеничні та крокуючі платформи. Проте колісні платформи мають такі переваги як вища швидкість переміщення по гладким поверхням та менші витрати енергії, а також плавність руху, простота конструкції та надійність. Оптимальні параметри конструкції колісного мобільного робота потрібно шукати для конкретних умов його експлуатації. Пришвидшити процес оптимізації можна шляхом комп'ютерної симуляції руху робота. Відомо чимало програмних продуктів для симуляції механіки роботів в двовимірному та тривимірному просторі (CoppeliaSim, Gazebo, Webots та інші). Як правило ці програми використовують такі рушії як ODE, Bullet, Chipmunk2D.

Авторами розроблено програму для оптимізації конструкції мобільних роботів з колісною платформою, які долають перешкоди у вигляді розсипаних на дорозі паралелепіпедів [27]. Критерієм оптимальності є найдовша відстань, що проїхав робот за відведений час. Було використано мову Python та її пакети PyODE 0.12, Scipy 1.14.1, VTK 5.8.0, odeViz 0.2.

Модуль ode4w_viz_example2.py використовує pyODE для симуляції руху мобільного робота. За допомогою v=True можна включити режим візуалізації на основі odeViz [28] та VTK. Модуль отримує значення параметрів конструкції робота через stdin. Якщо їх немає, то приймається значення за замовчуванням. Після цього програма створює об'єкт world (з гравітацією та лінійним і кутовим демпфуванням) і об'єкт space. Після цього створюється площина, по якій буде пересуватись робот. Перешкодами є розсипані на площині паралелепіпеди з вказаною густиною і розмірами. Кожен паралелепіпед створюється за допомогою функції create_box. В програмі створюється 100 таких об'єктів з випадковими координатами і кутами повороту. Після цього створюється корпус

робота у вигляді паралелепіпеда з вказаною густиною, розмірами і позицією. Далі створюється чотири колеса. Колеса можуть бути циліндричними або сферичними. Колеса також мають вказану густину і розміри. Після цього створюється з'єднання коліс з корпусом. Циліндричні колеса мають з'єднання HingeJoint з однією віссю обертання, а сферичні – Hinge2Joint з двома осями обертання. Також вказуються параметри підвіски. Для обробки контактів існує функція near_callback, яка автоматично викликається під час контакту. В ній установлюється такі параметри контакту, як пружність та коефіцієнт тертя. Контактні групи зберігаються в об'єкті contact_group. Для візуалізації створюється об'єкт viz і установлюється позиція камери. Після цього розпочинається цикл симуляції з кроком часу dt. Кількість ітерацій рівна 1000. В циклі вказується крутний момент на двох колесах, обробляється контакт, розраховується рух тіл, очищується contact_group і оновлюється зображення. Потім програма виводить в stdout кінцеву координату робота.

3.2. Модуль для оптимізації конструкції робота на основі еволюційного алгоритму

Модуль main.py використовує функцію differential_evolution з пакету SciPy для оптимізації конструкції робота на основі еволюційного алгоритму. Цей алгоритм доцільно застосовувати для великої кількості параметрів. Функція f(x) отримує значення параметрів конструкції робота, запускає в окремому процесі модуль ode4w_viz_example2.py, передає в його stdin значення параметрів конструкції та повертає кінцеву координату робота (від'ємне значення). Функція differential_evolution запускає процес оптимізації з параметрами maxiter=100, polish=False, workers=-1. Для прикладу оптимізували такі три параметри конструкції: радіус колеса, відстань між колесами по довжині і відстань між колесами по ширині. Границі значень цих параметрів вказані у списку bounds=[(0.1, 0.2), (0.4, 0.5), (0.3, 0.4)]. Для пришвидшення симуляції виконували розпаралелювання алгоритму за допомогою модуля multiprocessing. На мікропроцесорі Intel Xeon E5-2630L v3, який має 16 логічних процесорів,

обчислення тривало приблизно 20 хвилин. Найкращі значення параметрів x=[0.103, 0.496, 0.300]. В даному випадку результат неочевидний і колеса з меншим радіусом виявилися кращими. Проте це залежить від таких факторів коефіцієнт тертя, параметри підвіски, густина перешкод, коліс і корпусу і т.п.

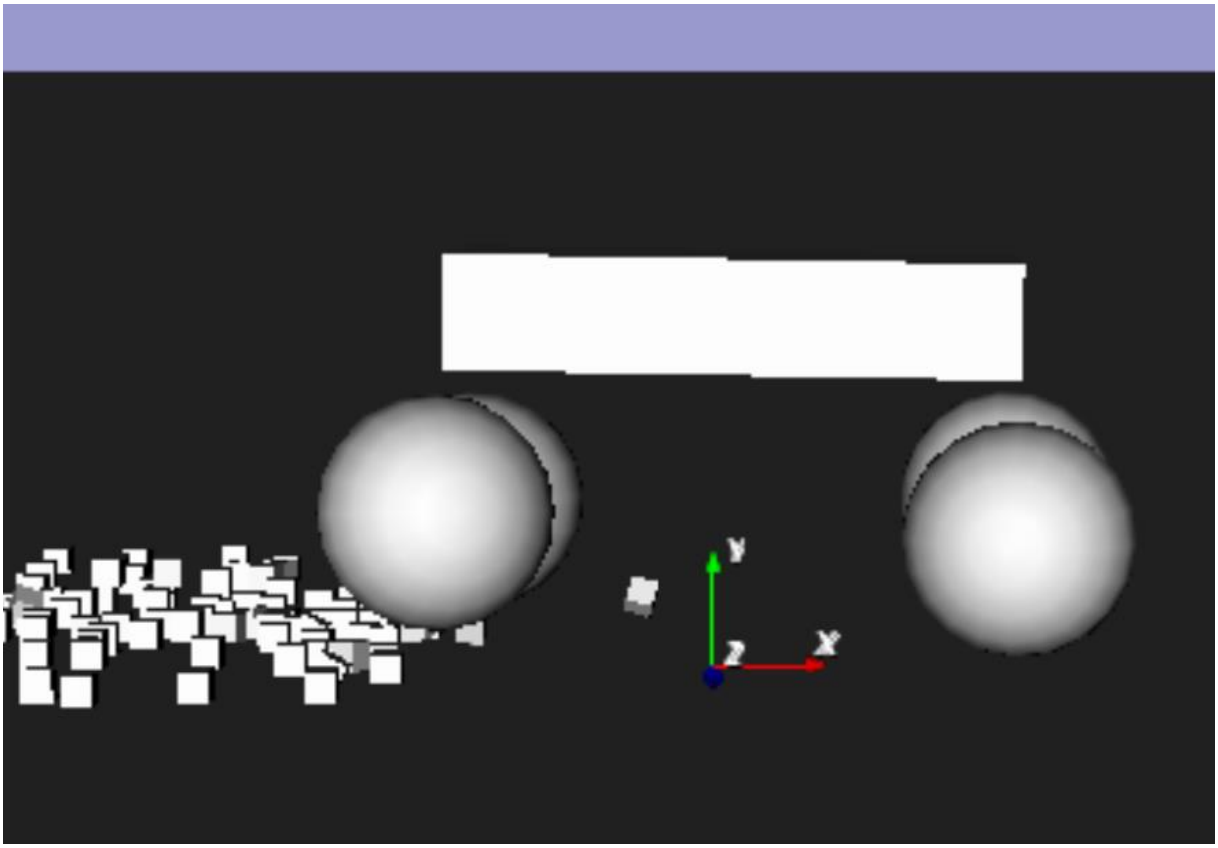


Рисунок 3.1 - Робот зі сферичними колесами долає перешкоди

Програма може бути використана для оптимізації більшої кількості параметрів, а також роботів, які мають іншу конструкцію. Також є можливість оптимізувати закон зміни крутних моментів на колесах під час подолання перешкод. Автори запрошують долучатись до цього проекту на GitHub.com щоб удосконалювати конструкції роботів та алгоритми оптимізації.

4 ПРОЕКТУВАННЯ МОБІЛЬНОГО РОБОТА

4.1 Параметрична модель робота

Параметричну модель робота створюємо за допомогою системи автоматизованого проєктування (САПР) FreeCAD, що використовується для проєктування, моделювання та оптимізації конструкцій на етапі розробки.

Проєктування конструкції робота починається зі створення основної платформи (рисунок 4.1), до якої у подальшому будуть кріпитися всі необхідні елементи та компоненти. Процес побудови цієї деталі є відносно простим і не потребує значних зусиль. Спочатку обирається одна з базових площин, на якій створюється ескіз із необхідними розмірами. До основних елементів ескізу належать пази для регулювання положення осей робота та прямокутні отвори для прокладки кабелів живлення і керування двигунами. Ці отвори й пази забезпечують функціональну адаптивність та зручність подальшої збірки конструкції.

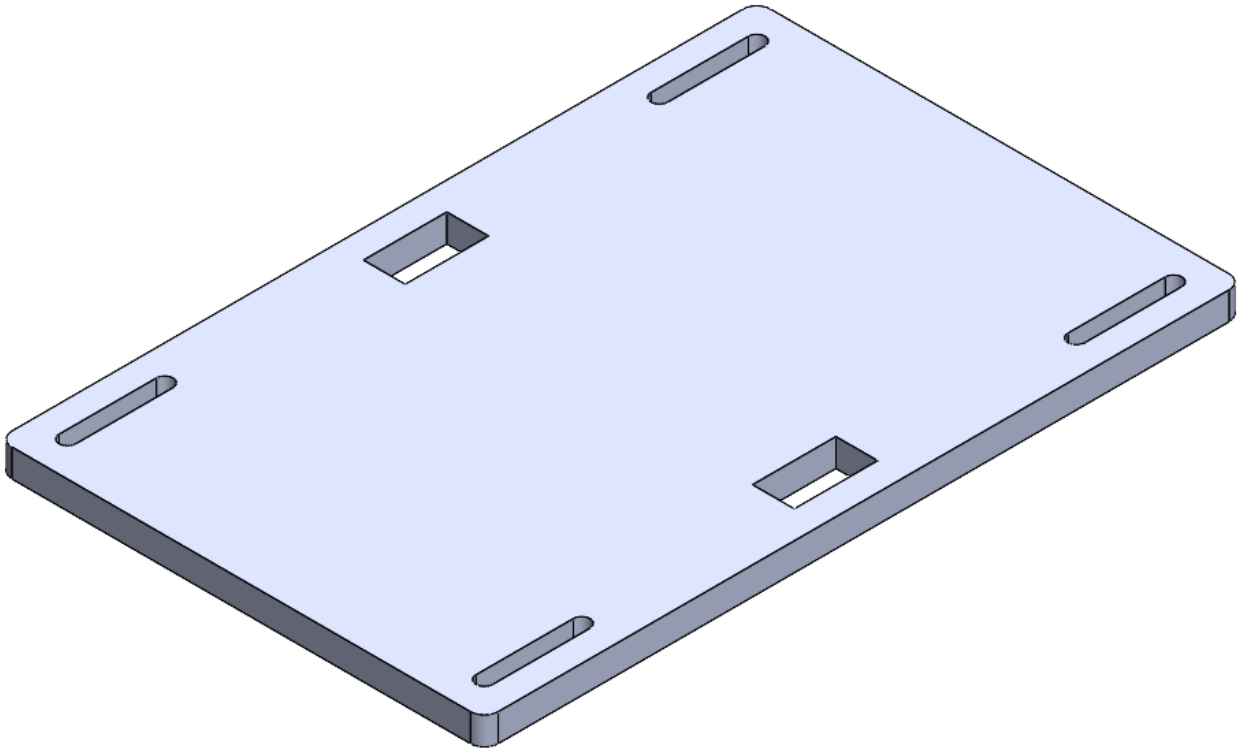


Рисунок 4.1 – Платформа робота

На рисунку 4.2 зображено 3-D модель деталі, що утримує сферичне колесо. Деталь представляє собою сферичну оболонку, яка за допомогою циліндра на верхній його поверхні кріпиться до деталі «Платформа робота». На сферичній оболонці передбаченно 6 місць із кронштейнами для кріплення шести омні-коліс для всенапрямленого руху. На дану деталь також кріпляться двигуни.

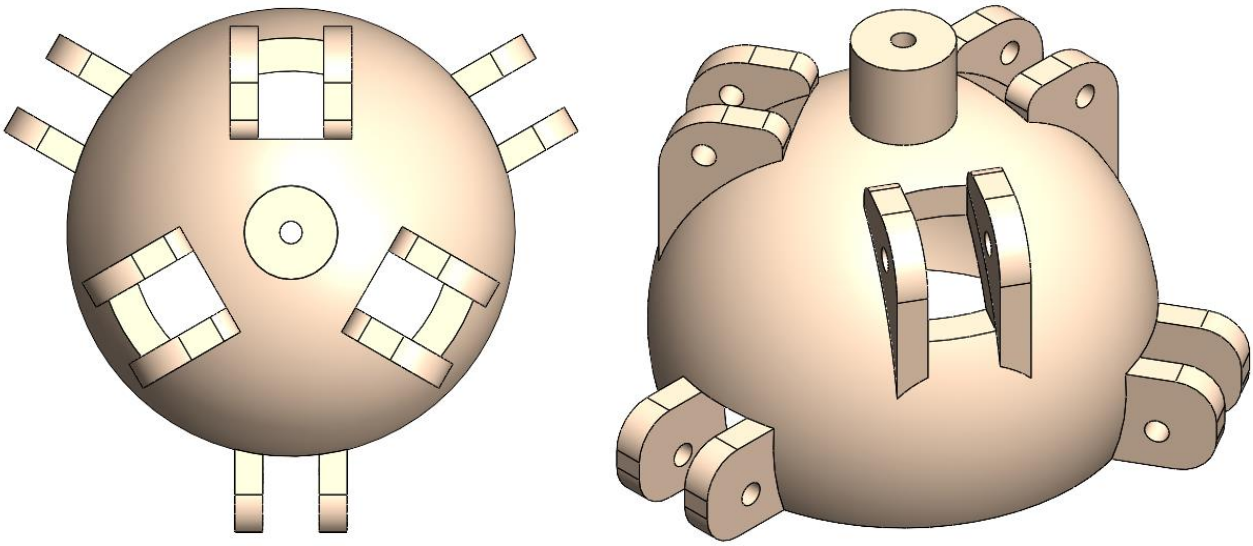


Рисунок 4.2 – 3-D модель деталі кріплення колеса

Оскільки двигун Servo DS3235 SG 35KG (рисунок 4.3) є стандартизованою деталлю, то його модель можна завантажити з відповідної бібліотеки чи знайти та завантажити готову модель з мережі Інтернет. Вибір впав саме на цей двигун тому, що він має можливість обертання на 360^{o}, що я є критично важливим у конструкції нашого колеса. За допомогою цих двигунів у рух приводяться тільки два всенапрямленні колеса інші чотири схугують підтримкою, та утримують сферу колеса у оболонці кріплення колеса.

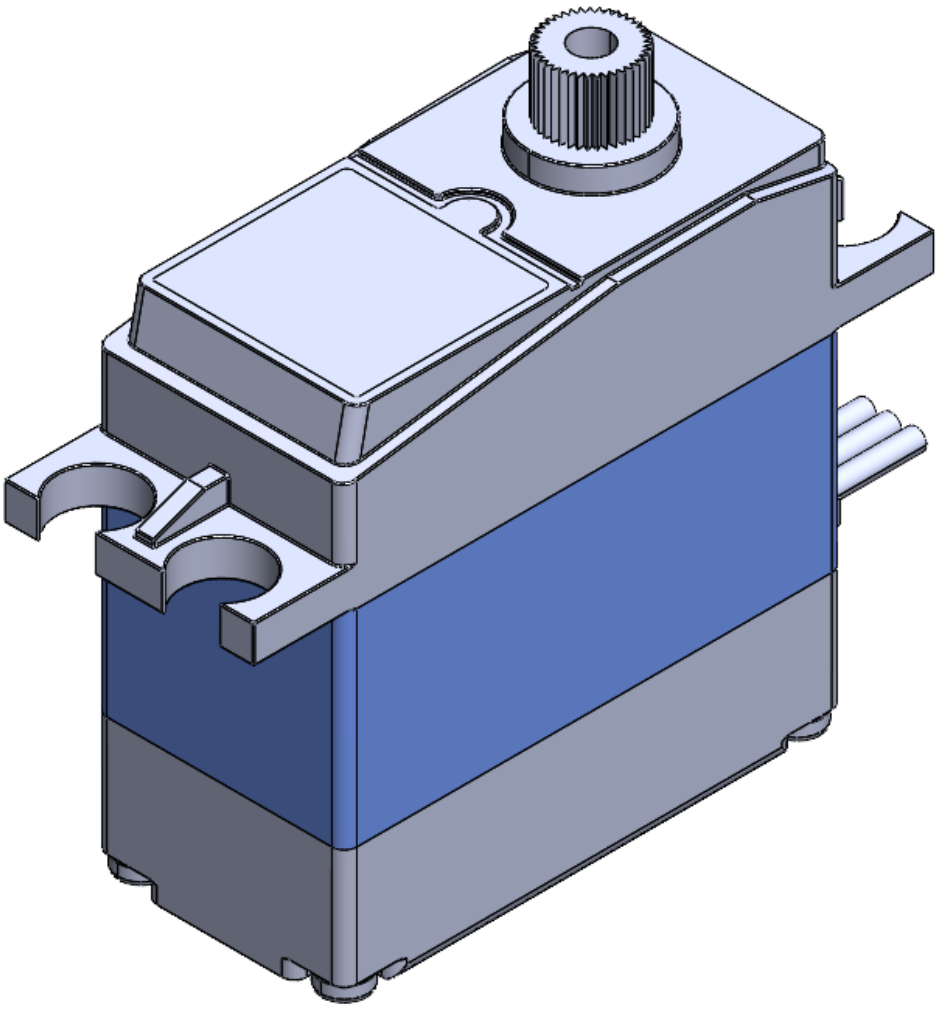


Рисунок 4.3 – Двигун Servo DS3235 SG 35KG

Модель колеса було розробляено самостійно, щоб забезпечити необхідній діаметр коліс згідно результатів оптимізації.

Моделювання колеса починаємо із вибору базової площини, на якій створюємо ескіз, задаючи необхідні нам розміри зовнішнього діаметру .За допомогою інструменту «обертання навкого осі» придаємо об’єм ескізу та отримуємо тверде тіло. Колесо складається із двох частин. Із м’якої ризинової оболонки та більш твердішої внутрішньої оболонки, яка надає потрібну жорсткість колесу (рисунок 4.4).

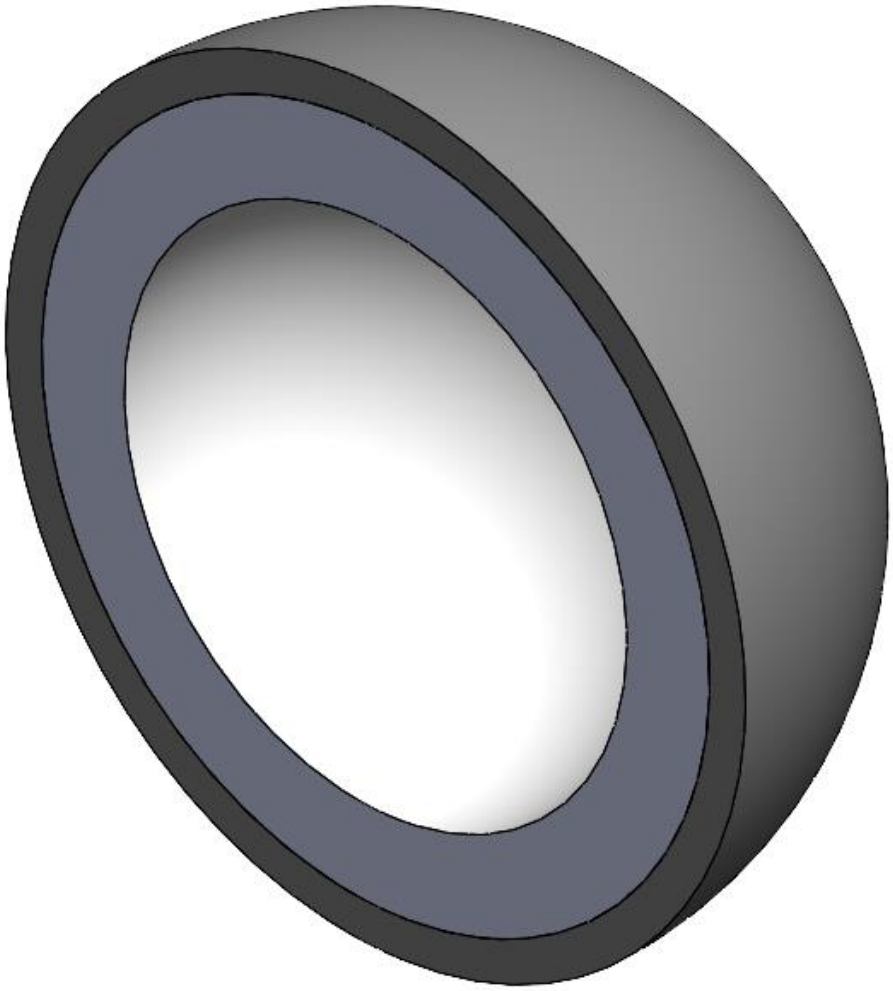


Рисунок 4.4 – Будова колеса

Моделі таких стандартизованих елементів як: Мікроконтролер ESP32 (рисунок 4.5), за допомогою якого і буде керуватися даний робот та батарейний блок 4x18650 (рисунок 4.6), для живлення, знаходимо у вільному доступі та завантажуємо.

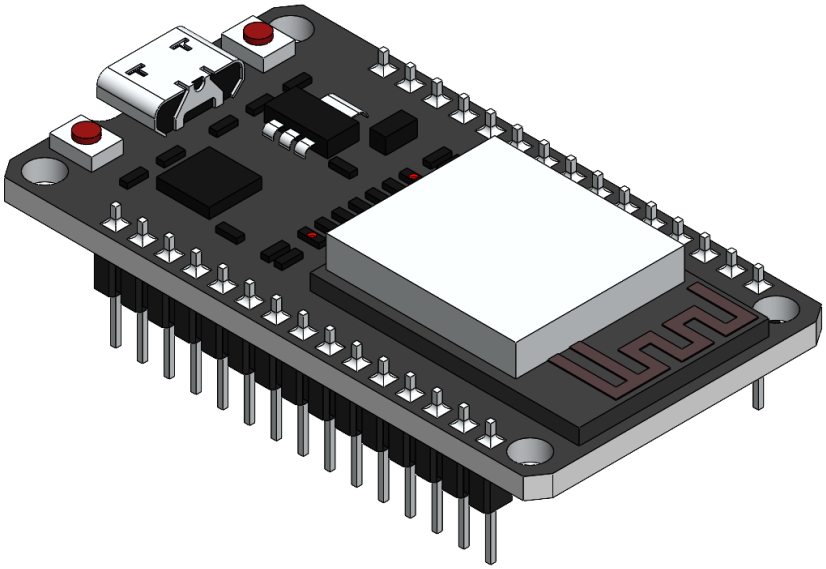


Рисунок 4.5 – Мікроконтролер ESP32

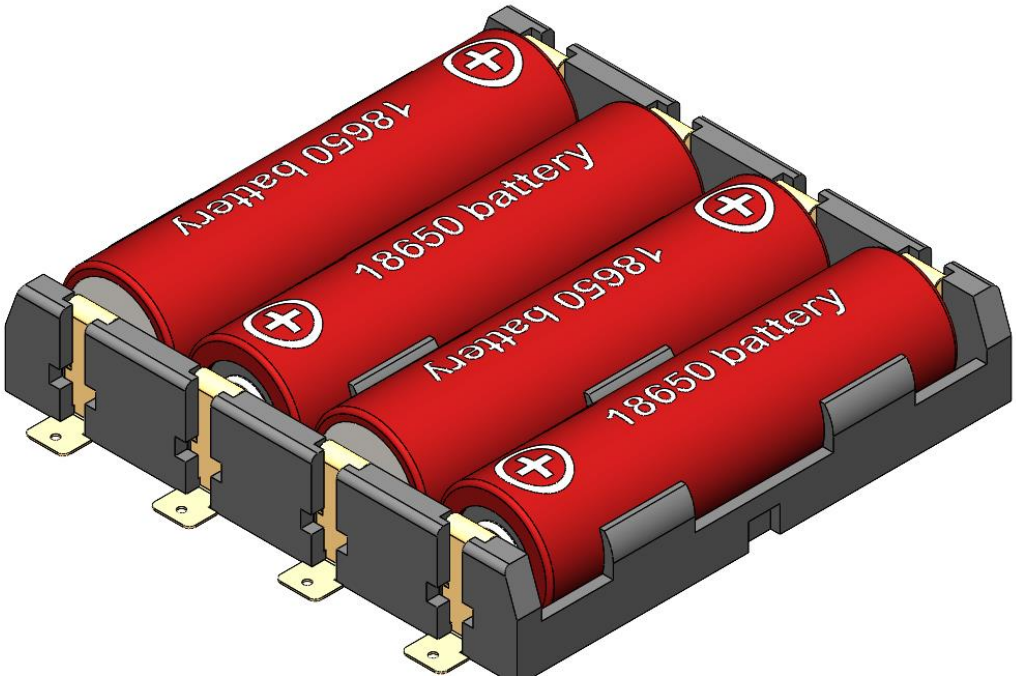


Рисунок 4.6 – Модель батарейного блоку 4x18650

Після моделювання усіх необхідних деталей, можна приступати до їх збірки у програмному середовищі FreeCAD. Для цього відкриваємо програму та вибираємо створити нову збірку. Після чого потрібно додати першу модель яка буде фіксованою в нашому випадку це буде платформа робота, адже усі наступні деталі будуть кріпитися уже на неї. Спершу за допомогою гвинтів прикріплюємо

кріплення коліс до платформи у якому згодом монтується саме сферичне колесо та двигуни із приводними омні-колесами та омні-колесами, що виконують функцію позиціонування та утримання сферичнорго колеса. До кріплення колеса також додаємо двигуни, що управляють колесами. Після чого залишається тільки додати, Мікроконтролер ESP32 та живлення для нього а саме два батарейні блоки 4x18650. На поверхн. Платформи монтуємо також два контейнера для перевезення користного вантажу. В результаті отримуємо готову 3D-збірку моделі робота (рисунок 4.7).

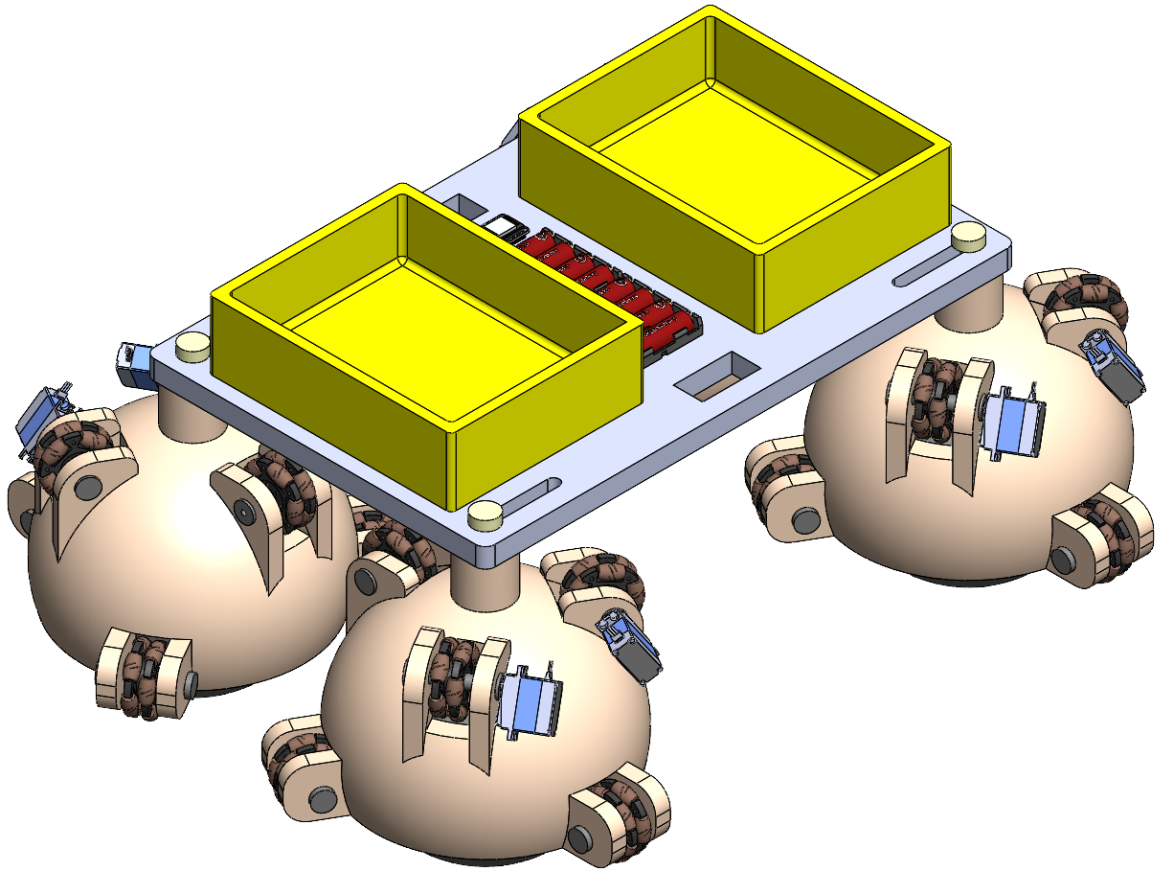


Рисунок 4.7 – Готова 3D-модель робота

4.2 Принципова електрична схема і керуюча програма робота

Після виконання моделі та її збірки було розробленно принципову електричну схему (рисунок 4.8), та програму для MicroPyton. Для керування роботом використовується 8 сервоприводів попарно розміщенних на кожному із коліс. Вони повинні мати здатність обертатись на 360 градусів.

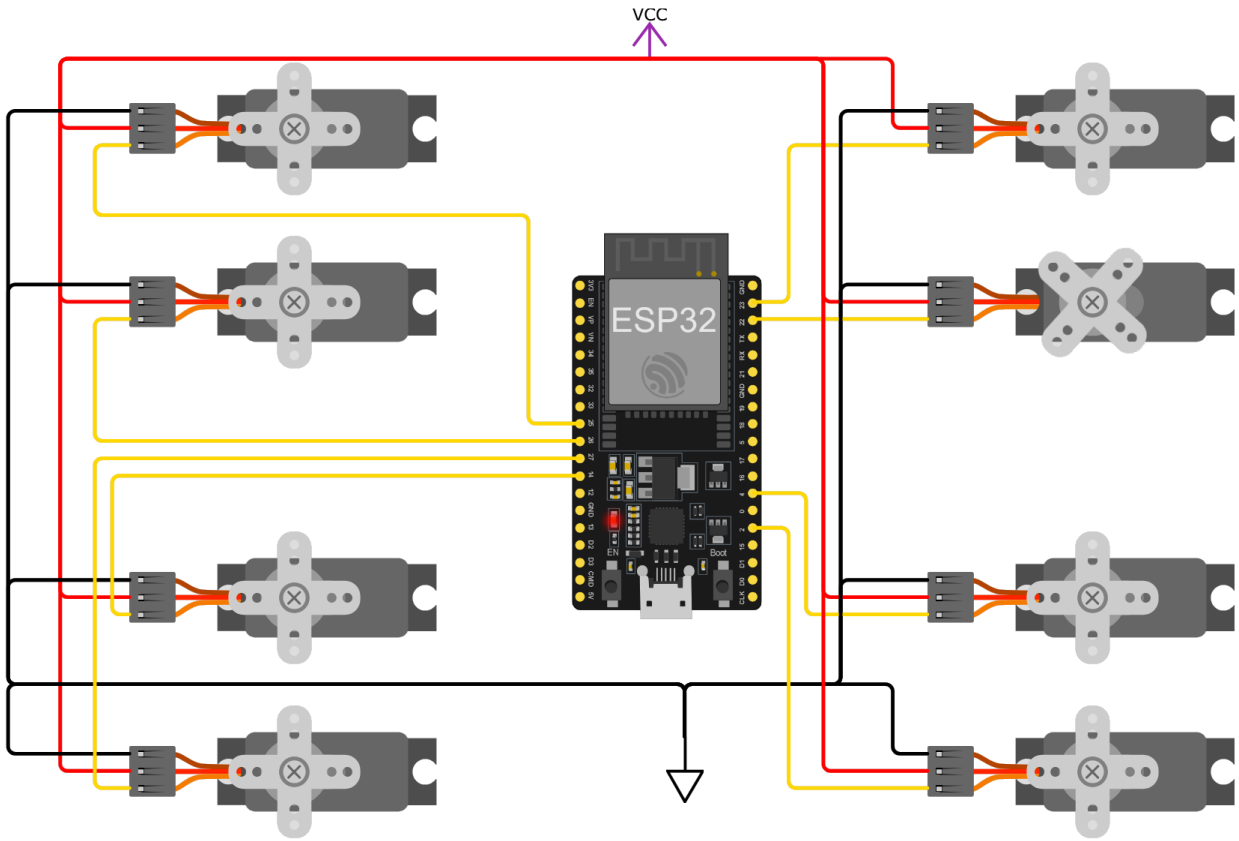


Рисунок 4.8 – Принципова електрична схема

За допомогою мови Python було розроблено програму для прошивки MicroPython мікроконтролера ESP32 (виконується на мікроконтролері). До цифрових пінів підключено вісім серврприводів, котрі мають можливість обертання на 360°. До прикладу SLB1 - це один із двох сервопривід, що розташований на лівому задньому колесі. На підключених пінах генерується широтноінмпульсна модуляція, тобто певний цифровий сигнал певної частоти від якого залежить швидкість обертання сервопривода. У нашому випадку після викликання функції run(s) – де «s» це конкретний об’єкт, наприклад (SLB1), то він він буде обертатися із заданою швидкістю. Після виклику функції stop(s) він зупиниться.

Після чого йде цикл для тестування цих функції, тобто після запуску run(sRF1) ми побачимо, що цей сервопривід обертається із заданою швидкістю. Також мікроконтролер ESP32 дає можливість під’єднатися Wi-Fi клієнту, це дає нам можливість передавати команди навіть зі смартфона. В потрійних лапках представлений цикл, який дає нам таку можливість.

```
from machine import Pin, PWM
from time import sleep

import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('192.168.1.116', 8888))
s.listen(1)

sRF1 = PWM(Pin(22), freq=50)
sRF2 = PWM(Pin(23), freq=50)

sRB1 = PWM(Pin(2), freq=50)
sRB2 = PWM(Pin(4), freq=50)

sLF1 = PWM(Pin(26), freq=50)
sLF2 = PWM(Pin(25), freq=50)

sLB1 = PWM(Pin(14), freq=50)
sLB2 = PWM(Pin(27), freq=50)

def run(s):
  s.duty(90) # обертання для серво 360
  sleep(0.1)
```

```
def stop(s):
  s.duty(50) # зупинка для серво 360
  sleep(0.1)

# тестування
while True:
  run(sRF1)
  #eval("run(sRF1)")
  sleep(1)
  stop(sRF1)
  sleep(1)

"""
# виконання команд від Wifi-клієнта
while True:
    soc, addr = s.accept()
    print('Server is connected to client', addr)
    command=soc.recv(255) # отримати команду
    print('Client:', command)
    eval(command) # виконати команду
    #eval("run(sRF1))")
    soc.close()
"""
```

ВИСНОВКИ

1. Проведено аналіз конструкцій мобільних роботів, у ході якого визначено, що перспективними для застосування є роботи на сферичних колесах, що і було взято за основу для виконання даної роботи.

2. Здійснено огляд методів оптимізації конструкцій роботизованих платформ і обґрунтовано доцільність використання еволюційних алгоритмів. Вони ефективні для задач з великою кількістю параметрів, випадковими перешкодами та оптимізацією конструкцій для підвищення продуктивності та надійності.

3. Розроблено програму для імітаційного моделювання переміщення мобільного робота та його оптимізації за допомогою еволюційних алгоритмів. У ході оптимізації вдалося досягти поліпшення параметрів конструкції, що підвищує стійкість і маневреність робота.

4. Спроєктовано параметричну модель мобільного робота в системі автоматизованого проєктування FreeCAD. В основу моделі покладено оптимальні значення параметрів, знайдені в процесі проведення імітаційних розрахунків.

5. Розроблено принципову електричну схему робота на основі доступних і недорогих компонентів, що включають мікроконтролер. Система керування передбачає, як автономну роботу так і дистанційне управління за допомогою команд.

6. Створена конструкція мобільного робота може бути використана для вирішення практичних завдань, таких як переміщення вантажів, обстеження важкодоступних чи небезпечних місць.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Промислова та мобільна робототехніка: URL: https://conf.ztu.edu.ua/wp-content/uploads/2020/05/6.-promyslova-ta-mobilna-robototehnika.pdf
2. Конференції Державного університету «Житомирська політехніка». URL: https://conf.ztu.edu.ua/wp-content/uploads/2020/05/6.-promyslova-ta-mobilna-robototehnika.pdf
3. 4WD Metal Smart Car Chassis. URL: https://www.amazon.ae/MC100-Omnidirectional-Platform-Raspberry-Learning/dp/B09V7CVHMK
4. X Series | Digital Concepts Engineering. Home | Digital Concepts Engineering - Robotic Control Systems. URL: https://dconcepts.co.uk/products/x-series/
5. Рефагі І. Дрон Х3 нового покоління: здатний вести розвідку, знаходити цілі та відволікати ворога. ФОКУС. URL: https://focus.ua/uk/digital/641106-dron-x3-novogo-pokolinnya-zdatniy-vesti-rozvidku-znahoditi-cili-ta-vidvolikati-voroga
6. Рефагі І. Дрон Х3 нового покоління: здатний вести розвідку, знаходити цілі та відволікати ворога. ФОКУС. URL: https://focus.ua/uk/digital/641106-dron-x3-novogo-pokolinnya-zdatniy-vesti-rozvidku-znahoditi-cili-ta-vidvolikati-voroga
7. Як дізнатися розміри гумової гусениці Що означає маркування на гусениці. Техноактив Інвест. Запчастини нові й уживані для будівельної та сільгосптехніки, спецтехніка. Прямі поставки з Європи будь-якої кількості. - Техноактив Інвест. URL: https://technoaktyv.com.ua/ua/a303782-kak-pravilno-opredelit.html
8. Tracks military technology, Tracked robots uses, advantages and disadvantages | Science online. Science online. URL: https://www.online-sciences.com/robotics/tracks-military-technology-tracked-robots-uses-advantages-disadvantages/
9. Tracked Robot Chassis: A Versatile and Powerful Platform. URL: https://www.prestigiouspress.com/tracked-robot-chassis-a-versatile-and-powerful-platform/
10. Харківський національний університет радіоелектроніки кафедра комп'ютерно-інтегрованих технологій, автоматизації та мехатроніки (КІТАМ) ЗБІРНИК студентських наукових статей «Автоматизація та приладобудування» «Automation and Development of Electronic Devices» ADED-2019 (Випуск 2) [електронне видання] Харків 2019
11. Електронний архів Харківського національного університету радіоелектроніки. URL: https://openarchive.nure.ua/home (access 20.06.23
12. Spot | Boston Dynamics. Boston Dynamics. URL: https://bostondynamics.com/products/spot/
13. Robotics Aug 2022. arXiv.orge-Printarchive. URL: https://arxiv.org/list/cs.RO/2022-08

14. OmniWheg: A transformable robot system with omnidirectional wheel legs - Kerosene Lamp Technology. Kerosene Lamp Technology - Embrace high technology and enjoy the future together. URL: https://www.victorlamp.com/article/7388621604
15. Fadelli I. A transformable robot with an omnidirectional wheel-leg. Tech Xplore - Technology and Engineering news. URL: https://techxplore.com/news/2022-12-robot-omnidirectional-wheel-leg.html (date of access: 16.12.2024).
16. Goodyear показала Eagle 360 Urban – розумні сферичні шини майбутнього (відео). URL: https://ecotechnica.com.ua/transport/goodyear-pokazala-eagle-360-urban-umnye-sfericheskie-shiny-budushchego-video
17. Сферичні шини Goodyear Eagle 360 навчили “спілкуватися і лікувати проколи” | ВІКНА. Новини Калуша та Прикарпаття. Vikna. URL: https://vikna.if.ua/cikavo/69625/view
18. Goodyear EMEA. Goodyear Unveils the Eagle 360 Urban, a Concept Tire Powered by Artificial Intelligence. Goodyear Unveils the Eagle 360 Urban, a Concept Tire Powered by Artificial Intelligence. URL: https://news.goodyear.eu/goodyear-unveils-the-eagle-360-urban-a-concept-tire-powered-by-artificial-intelligence/
19. Spherical Drive System. URL: https://www.facebook.com/p/Spherical-Drive-System-100064033755038/
20. US8459383B1 - Spherical drive system - Google Patents. Google Translate. URL: https://patents-google-com.translate.goog/patent/US8459383B1/en?_x_tr_sl=en&_x_tr_tl=uk&_x_tr_hl=uk&_x_tr_pto=sc&_x_tr_hist=true
21. Дерзська О. Правосуддя на ходу: китайський робот-поліцейський може ловити злочинців "стильно". ФОКУС. URL: https://focus.ua/uk/digital/683310-u-kitaji-sferichniy-robot-lovit-zlochinciv-zavdyaki-shi
22. Spherical robot: A novel robot for exploration in harsh unknown environments. https://ietresearch.onlinelibrary.wiley.com/. URL: https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/csy2.12099.
23. A Review of Physics Simulators for Robotic Applications / Дж. Коллінз et al. IEEE Xplore. URL: https://ieeexplore.ieee.org/document/9386154
24. What is the best simulation tool for robotics? - Robohub. Robohub - Connecting the robotics community to the world. URL: https://robohub.org/what-is-the-best-simulation-tool-for-robotics/
25. Robot simulator CoppeliaSim: create, compose, simulate, any robot - Coppelia Robotics. Robot simulator CoppeliaSim: create, compose, simulate, any robot - Coppelia Robotics. URL: https://www.coppeliarobotics.com/
26. Open Dynamics Engine. Open Dynamics Engine. URL: https://www.ode.org/
27. pyODE_examples https://github.com/vkopey/pyODE_examples (accessed 10.12.2024)
28. odeViz https://github.com/andre-dietrich/odeViz (accessed 10.12.2024)

ДОДАТКИ

Додаток А – Програма для оптимізації мобільного робота

```
import subprocess
import multiprocessing
import numpy as np
from scipy.optimize import differential_evolution

def f(x):
    p='d:\\Portable\\PortablePython27VTK_ODE\\App\\Python\\'
    command = [p+'python.exe', p+'ode4w_viz_example2.py']
    process = subprocess.Popen(command, stdin=subprocess.PIPE,
stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
    stdout, stderr = process.communicate(input=np.array2string(x))
    return float(stdout)

if __name__=='__main__':
    #print(f(np.array([0.1, 0.5, 0.3])))

    multiprocessing.freeze_support()
    bounds = [(0.1, 0.2), (0.4, 0.5), (0.3, 0.4)]
    result = differential_evolution(f, bounds, maxiter=100, polish=False, workers=-1)
    print(result.x)
    print(result.fun)
```

Додаток Б – Програма для симуляції мобільного робота

```
from numpy import fromstring
import ode
v=0
if v:
    import odeViz.ode_visualization as ode_viz
import sys, random

x=sys.stdin.read()
if x:
    x=x[1:-1]
    x=fromstring(x, dtype=float, sep=' ')
else:
    x=[0.10334363, 0.49620229, 0.30048395] #-8.28981301768

world = ode.World()
world.setGravity((0, -9.81, 0))
world.setLinearDamping(0.1)
world.setAngularDamping(0.1)

space = ode.Space()

ground = ode.GeomPlane(space, (0, 1, 0), 0)

def create_box(world, space, density, lx, ly, lz):
    body = ode.Body(world)
    M = ode.Mass()
    M.setBox(density, lx, ly, lz)
    body.setMass(M)
    body.shape = "box"
```

```
    body.boxsize = (lx, ly, lz)
    # Create a box geom for collision detection
    geom = ode.GeomBox(space, lengths=body.boxsize)
    geom.setBody(body)
    return body, geom

B,G=[],[]
for i in range(100):
    b,g=create_box(world, space, 700, 0.05,0.05,0.05)
    b.setPosition((random.uniform(-2,-1),random.uniform(0,0.5),random.uniform(-
1,1)))
    b.setRotation([1,random.uniform(0,3),0, random.uniform(0,3),1,0, 0,0,1])
    B.append(b)
    G.append(g)

chassis = ode.Body(world)
M = ode.Mass()
M.setBox(10, 1, 0.2, 0.5)  # density, lx, ly, lz
chassis.setMass(M)
chassis.setPosition((0, 0.5, 0))

chassis_geom = ode.GeomBox(space, lengths=(1, 0.2, 0.5))
chassis_geom.setBody(chassis)

wheels = []
wheel_geoms = []
#wheel_positions = [(-0.5, 0, 0.3), (0.5, 0, 0.3), (-0.5, 0, -0.3), (0.5, 0, -0.3)]
wheel_positions = [(-x[1], 0, x[2]), (x[1], 0, x[2]), (-x[1], 0, -x[2]), (x[1], 0, -x[2])]
for pos in wheel_positions:
    wheel = ode.Body(world)
```

```
    M = ode.Mass()
    M.setSphere(1, x[0])  # density=1, radius=0.2
    #M.setCylinderTotal(0.1, 1, x[0], 0.05)
    wheel.setMass(M)
    wheel.setPosition((pos[0], pos[1] + 0.1, pos[2]))
    wheels.append(wheel)

    wheel_geom = ode.GeomSphere(space, radius=x[0]) #0.2
    #wheel_geom = ode.GeomCylinder(space, radius=x[0], length=0.05)
    wheel_geom.setBody(wheel)
    wheel_geoms.append(wheel_geom)

# Create joints to connect wheels to the chassis
joints = []
for i, wheel in enumerate(wheels):
    #joint = ode.HingeJoint(world)
    joint = ode.Hinge2Joint(world)
    joint.attach(chassis, wheel)
    joint.setAnchor(wheel.getPosition())
    #joint.setAxis((0, 0, 1))
    joint.setAxis1((0, 1, 0))
    joint.setAxis2((0, 0, 1))
    joint.setParam(ode.ParamSuspensionERP, 0.5)
    joint.setParam(ode.ParamSuspensionCFM, 0.8)
    joints.append(joint)

contact_group = ode.JointGroup()
# Collision callback function
def near_callback(args, geom1, geom2):
    # Check for collisions
```

```
    contacts = ode.collide(geom1, geom2)
    for contact in contacts:
        contact.setBounce(0.8)
        contact.setMu(5)
        joint = ode.ContactJoint(world, contact_group, contact)
        joint.attach(geom1.getBody(), geom2.getBody())

# Initialize the visualization
if v:
    viz = ode_viz.ODE_Visualization(world, [space], dt = 0.01)
    viz.GetActiveCamera().SetPosition(0,1,20)
    viz.update()

# Simulation loop
dt = 0.01
for i in range(1000):
    wheels[0].addTorque((0,0,2))
    wheels[2].addTorque((0,0,2))
    #joints[0].addTorque(-1)
    #joints[2].addTorque(-1)
    #wheels[1].setAngularVel((0,0,20))
    #wheels[3].setAngularVel((0,0,20))
    space.collide((world, contact_group), near_callback)
    world.step(dt)
    contact_group.empty()
    if v: viz.update()

#print np.array2string(x)
print chassis.getPosition()[0]
exit()
```

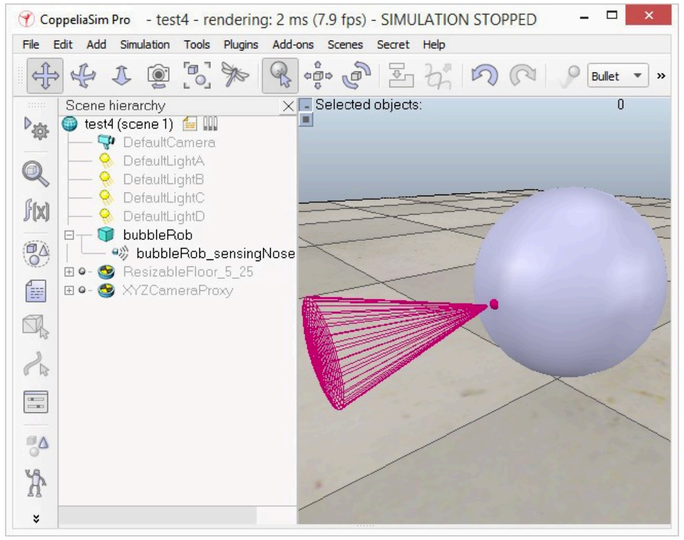


Рисунок 1 - Етап побудови моделі 1

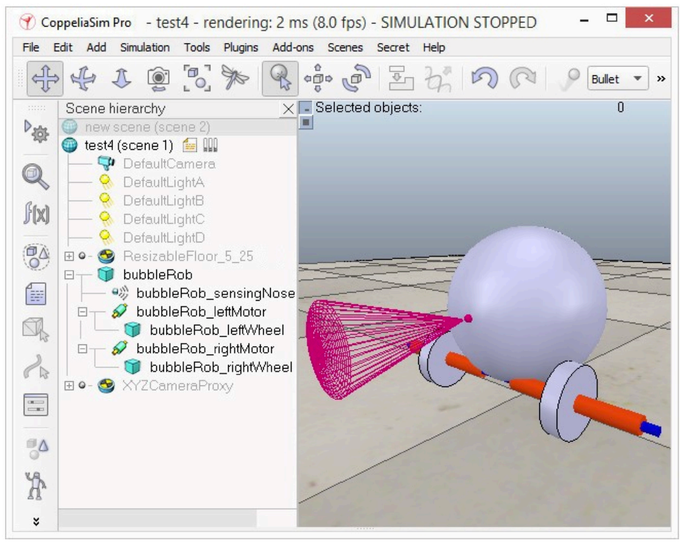


Рисунок 2 - Етап побудови моделі 2

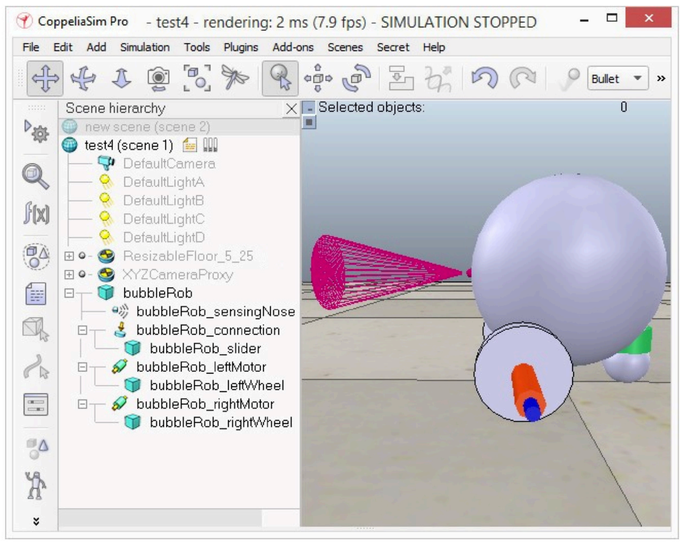


Рисунок 3 - Етап побудови моделі 3

```
#python

def sysCall_init():
    sim = require('sim')
    simVision = require('simVision')

def sysCall_vision(inData):
    simVision.sensorImgToWorkImg(inData['handle']) #скопіювати зображення датчика зору в робоче зображення
    simVision.edgeDetectionOnWorkImg(inData['handle'], 0.2) # виконати визначення країв на робочому зображенні
    simVision.workImgToSensorImg(inData['handle']) # копіювати робоче зображення в буфер зображення датчика зору

#python

import math

def sysCall_init():
    # Це виконується рівно один раз, коли цей сценарій виконується вперше
    sim = require('sim')
    simUI = require('simUI')
    self.bubbleRobBase = sim.getObject('..') # Це дескриптор bubbleRob
    self.leftMotor = sim.getObject("../leftMotor") # Ручка лівого двигуна
    self.rightMotor = sim.getObject("../rightMotor") # Ручка правого двигуна
    self.noseSensor = sim.getObject("../sensingNose") # Ручка датчика наближення
    self.minMaxSpeed = [50*math.pi/180, 300*math.pi/180] # Мінімальна та максимальна швидкість для кожного двигуна
    self.backUntilTime = -1 # Повідомляє, чи знаходиться bubbleRob у прямому чи зворотному режимі
    self.robotCollection = sim.createCollection(0)
    sim.addItemToCollection(self.robotCollection, sim.handle_tree, self.bubbleRobBase, 0)
    self.distanceSegment = sim.addDrawingObject(sim.drawing_lines, 4, 0, -1, 1, [0, 1, 0])
    self.robotTrace = sim.addDrawingObject(sim.drawing_linestrip + sim.drawing_cyclic, 2, 0, -1, 200, [1, 1, 0], None, None, [1, 1, 0])
    self.graph = sim.getObject('../graph')
    # sim.destroyGraphCurve(self.graph, -1)
    self.distStream = sim.addGraphStream(self.graph, 'bubbleRob clearance', 'm', 0, [1, 0, 0])
    # Створення власного інтерфейсу користувача UI:
    xml = '<ui title="' + sim.getObjectAlias(self.bubbleRobBase, 1) + ' speed" closeable="false" resizable="false" activate="false">'
    xml = xml + '<hslider minimum="0" maximum="100" on-change="speedChange_callback" id="1"/>'
    xml = xml + '<label text="" style="* {margin-left: 300px;}"></label></ui>'
    self.ui = simUI.create(xml)
    self.speed = (self.minMaxSpeed[0] + self.minMaxSpeed[1]) * 0.5
    simUI.setSliderValue(self.ui, 1, 100 * (self.speed - self.minMaxSpeed[0]) / (self.minMaxSpeed[1] - self.minMaxSpeed[0]))

def sysCall_sensing():
    result, distData, *_ = sim.checkDistance(self.robotCollection, sim.handle_all)
    if result > 0:
        sim.addDrawingObjectItem(self.distanceSegment, None)
        sim.addDrawingObjectItem(self.distanceSegment, distData)
        sim.setGraphStreamValue(self.graph,self.distStream, distData[6])
    p = sim.getObjectPosition(self.bubbleRobBase)
    sim.addDrawingObjectItem(self.robotTrace, p)

def speedChange_callback(ui, id, newVal):
    self.speed = self.minMaxSpeed[0] + (self.minMaxSpeed[1] - self.minMaxSpeed[0]) * newVal / 100

def sysCall_actuation():
    result, *_ = sim.readProximitySensor(self.noseSensor) # Зчитування датчика наближення
    # Якщо ми щось виявили, ми встановлюємо зворотний режим:
 якщо результат > 0:
        self.backUntilTime = sim.getSimulationTime() + 4
    if self.backUntilTime < sim.getSimulationTime():
        # У режимі вперед ми просто рухаємося вперед із бажаною швидкістю
        sim.setJointTargetVelocity(self.leftMotor, self.speed)
        sim.setJointTargetVelocity(self.rightMotor, self.speed)
    else:
        # У зворотному режимі ми просто виконуємо резервне копіювання за кривою зі зниженою швидкістю
        sim.setJointTargetVelocity(self.leftMotor, -self.speed / 2)
        sim.setJointTargetVelocity(self.rightMotor, -self.speed / 8)

def sysCall_cleanup():
    simUI.destroy(self.ui)
```

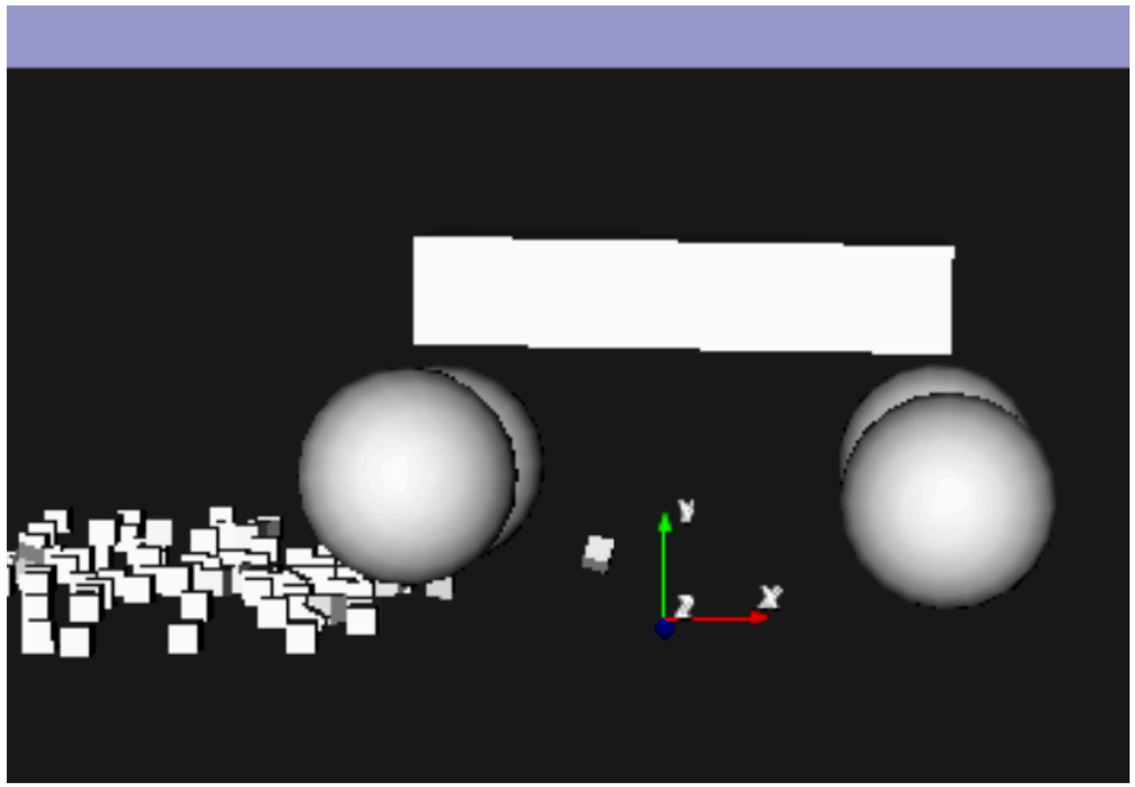


Рисунок 1 - Робот зі сферичними колесами долає перешкоди

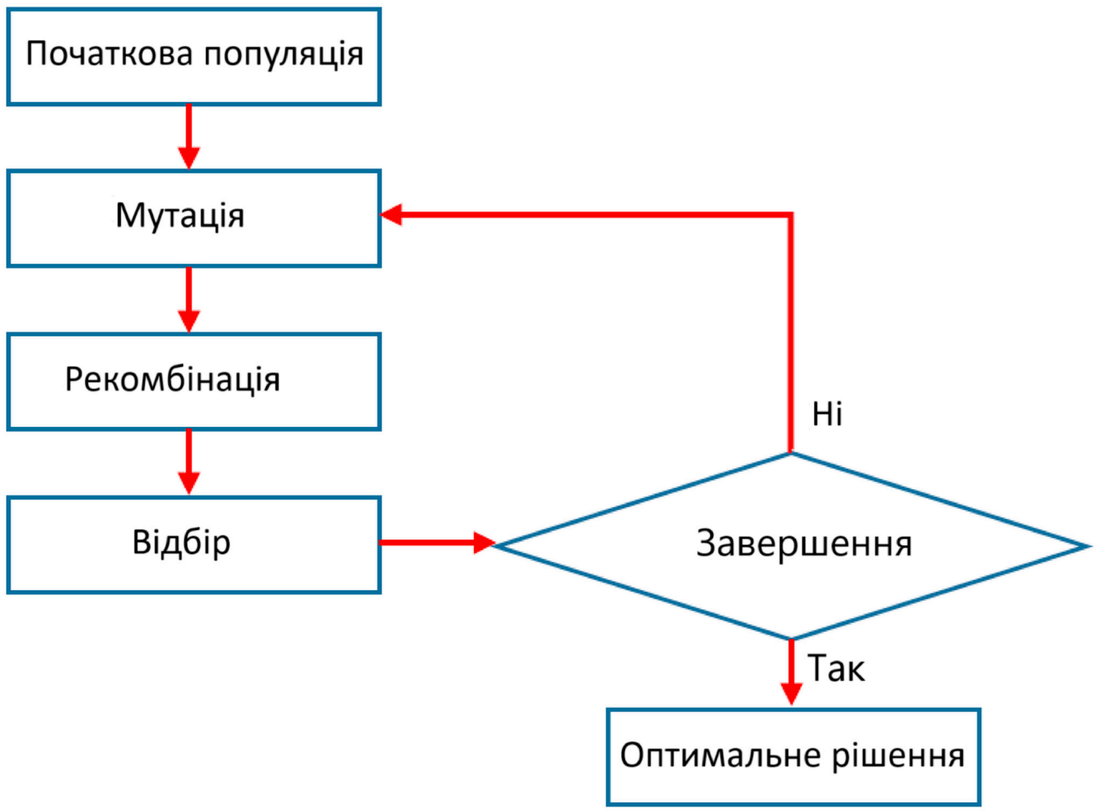
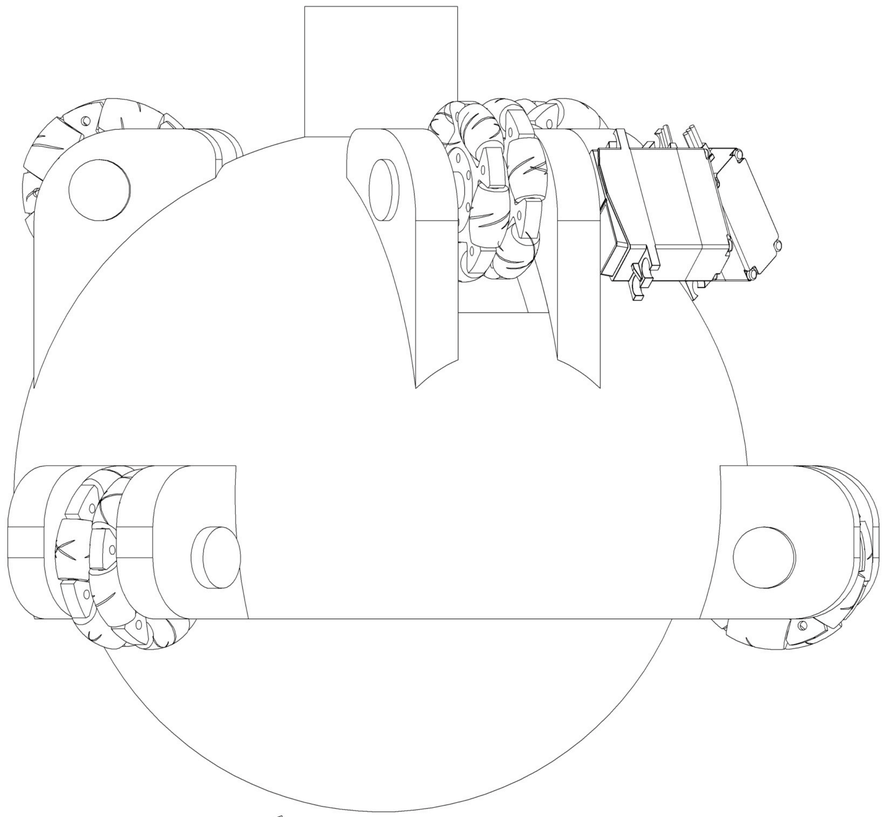


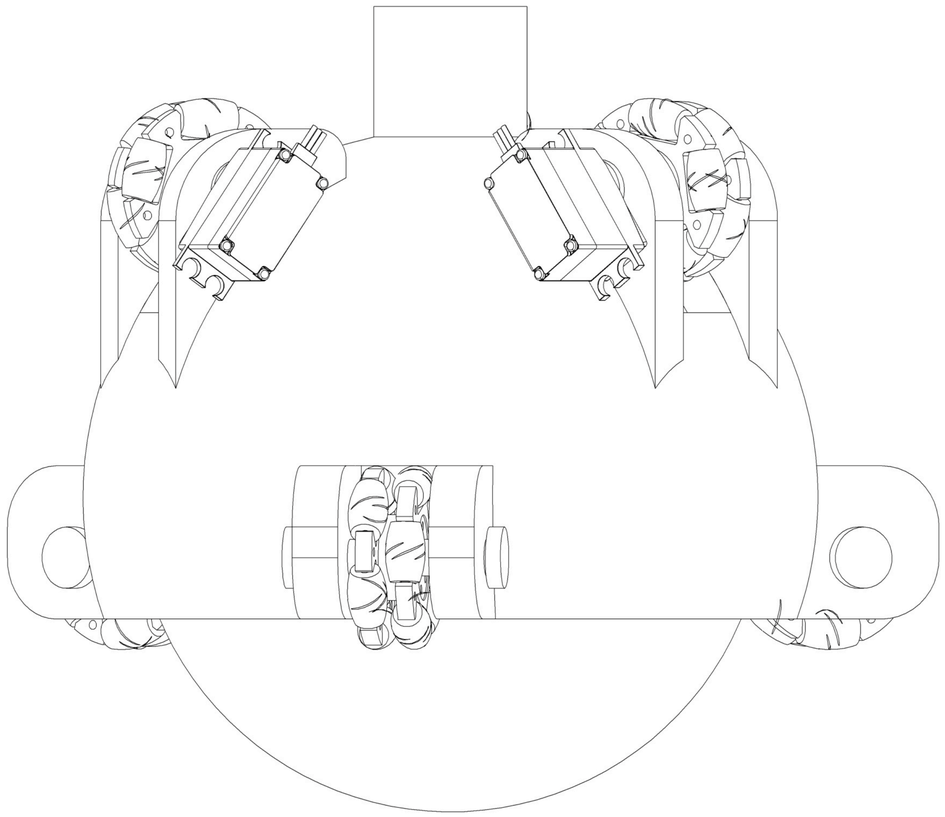
Рисунок 2 - Алгоритм диференціальної еволюції

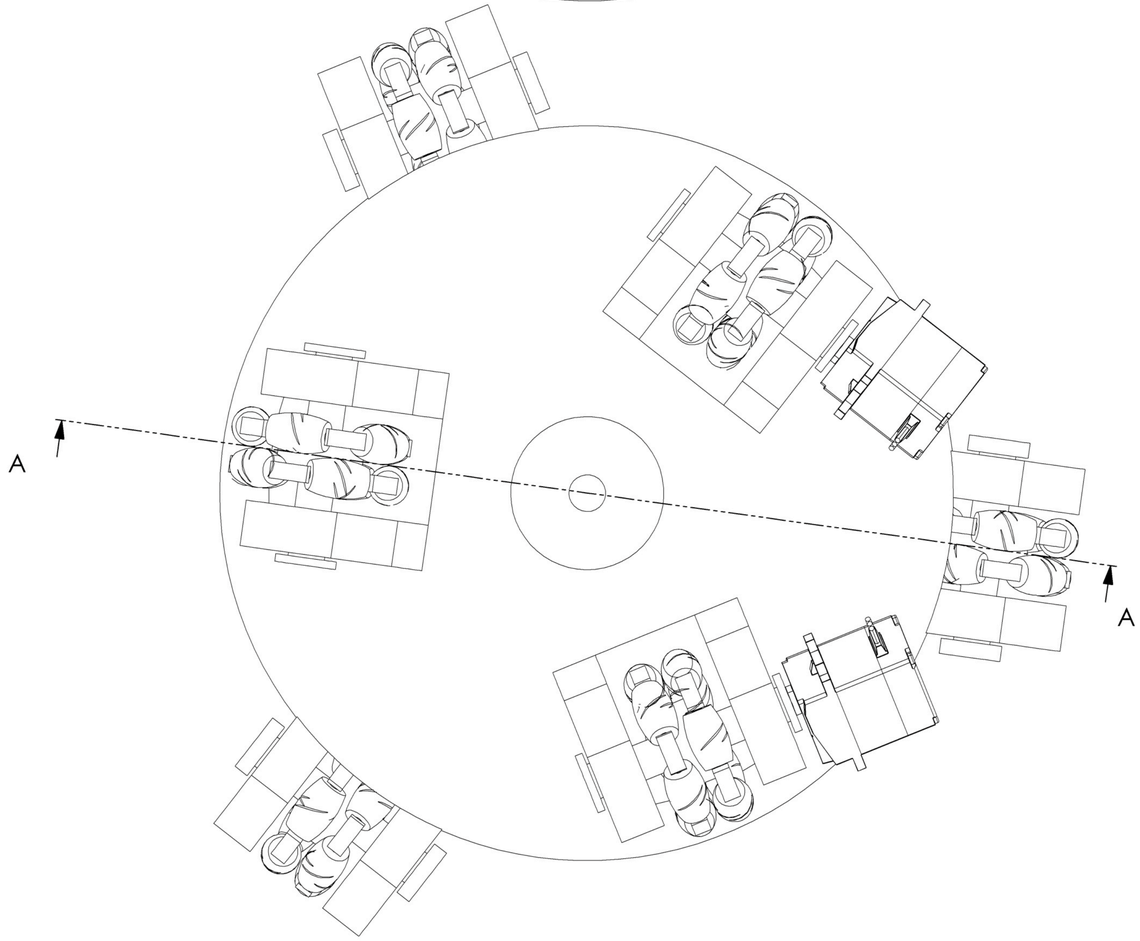
```
from numpy import fromstring
import ode
v=0
if v:
    import odeViz.ode_visualization as ode_viz
import sys, random

x=sys.stdin.read()
if x:
    x=x[1:-1]
    x=fromstring(x, dtype=float, sep=' ')
else:
    x=[0.10334363, 0.49620229, 0.30048395] #-8.28981301768

world = ode.World()
world.setGravity((0, -9.81, 0))
world.setLinearDamping(0.1)
world.setAngularDamping(0.1)

space = ode.Space()

ground = ode.GeomPlane(space, (0, 1, 0), 0)

def create_box(world, space, density, lx, ly, lz):
    body = ode.Body(world)
    M = ode.Mass()
    M.setBox(density, lx, ly, lz)
    body.setMass(M)
    body.shape = "box"
    body.boxsize = (lx, ly, lz)
    # Create a box geom for collision detection
    geom = ode.GeomBox(space, lengths=body.boxsize)
    geom.setBody(body)
    return body, geom

B,G=[],[]
for i in range(100):
    b,g=create_box(world, space, 700, 0.05,0.05,0.05)
    b.setPosition((random.uniform(-2,-1),random.uniform(0,0.5),
random.uniform(-1,1)))
    b.setRotation([1,random.uniform(0,3),0, random.uniform(0,3),1,0, 0,0,1])
    B.append(b)
    G.append(g)

chassis = ode.Body(world)
M = ode.Mass()
M.setBox(10, 1, 0.2, 0.5)  # density, lx, ly, lz
chassis.setMass(M)
chassis.setPosition((0, 0.5, 0))

chassis_geom = ode.GeomBox(space, lengths=(1, 0.2, 0.5))
chassis_geom.setBody(chassis)

wheels = []
wheel_geoms = []
#wheel_positions = [(-0.5, 0, 0.3), (0.5, 0, 0.3), (-0.5, 0, -0.3), (0.5, 0, -0.3)]
wheel_positions = [(-x[1], 0, x[2]), (x[1], 0, x[2]), (-x[1], 0, -x[2]), (x[1], 0, -
x[2])]
for pos in wheel_positions:
    wheel = ode.Body(world)
    M = ode.Mass()
    M.setSphere(1, x[0])  # density=1, radius=0.2
    #M.setCylinderTotal(0.1, 1, x[0], 0.05)
    wheel.setMass(M)
    wheel.setPosition((pos[0], pos[1] + 0.1, pos[2]))
    wheels.append(wheel)

    wheel_geom = ode.GeomSphere(space, radius=x[0]) #0.2
    #wheel_geom = ode.GeomCylinder(space, radius=x[0], length=0.05)
    wheel_geom.setBody(wheel)
    wheel_geoms.append(wheel_geom)

# Create joints to connect wheels to the chassis
joints = []
for i, wheel in enumerate(wheels):
    #joint = ode.HingeJoint(world)
    joint = ode.Hinge2Joint(world)
    joint.attach(chassis, wheel)
    joint.setAnchor(wheel.getPosition())
    #joint.setAxis((0, 0, 1))
    joint.setAxis1((0, 1, 0))
    joint.setAxis2((0, 0, 1))
    joint.setParam(ode.ParamSuspensionERP, 0.5)
    joint.setParam(ode.ParamSuspensionCFM, 0.8)
    joints.append(joint)

contact_group = ode.JointGroup()
# Collision callback function
def near_callback(args, geom1, geom2):
    # Check for collisions
    contacts = ode.collide(geom1, geom2)
    for contact in contacts:
        contact.setBounce(0.8)
        contact.setMu(5)
        joint = ode.ContactJoint(world, contact_group, contact)
        joint.attach(geom1.getBody(), geom2.getBody())

# Initialize the visualization
if v:
    viz = ode_viz.ODE_Visualization(world, [space], dt = 0.01)
    viz.GetActiveCamera().SetPosition(0,1,20)
    viz.update()

# Simulation loop
dt = 0.01
for i in range(1000):
    wheels[0].addTorque((0,0,2))
    wheels[2].addTorque((0,0,2))
    #joints[0].addTorque(-1)
    #joints[2].addTorque(-1)
    #wheels[1].setAngularVel((0,0,20))
    #wheels[3].setAngularVel((0,0,20))
    space.collide((world, contact_group), near_callback)
    world.step(dt)
    contact_group.empty()
    if v: viz.update()

#print np.array2string(x)
print chassis.getPosition()[0]
exit()
```

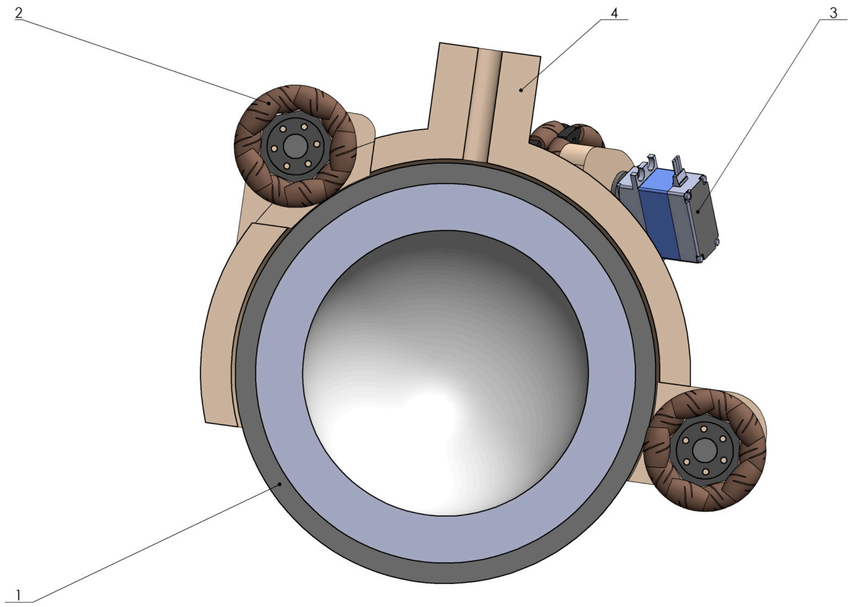
						МР.ПМКм-31.00.00.000	
Зм.	Арк.	№ докум.	Підп.	Дата			
Розроб.		Корбеляк Р.В.			Програма симуляції	Літ.	Маса	Масштаб
Перев.		Копей В.Б.						
Т. контр.						Арк.	Аркушів	
Н. контр.						ІФНТУНГ ПМКм-23-1	
Затв.							

Формат А1



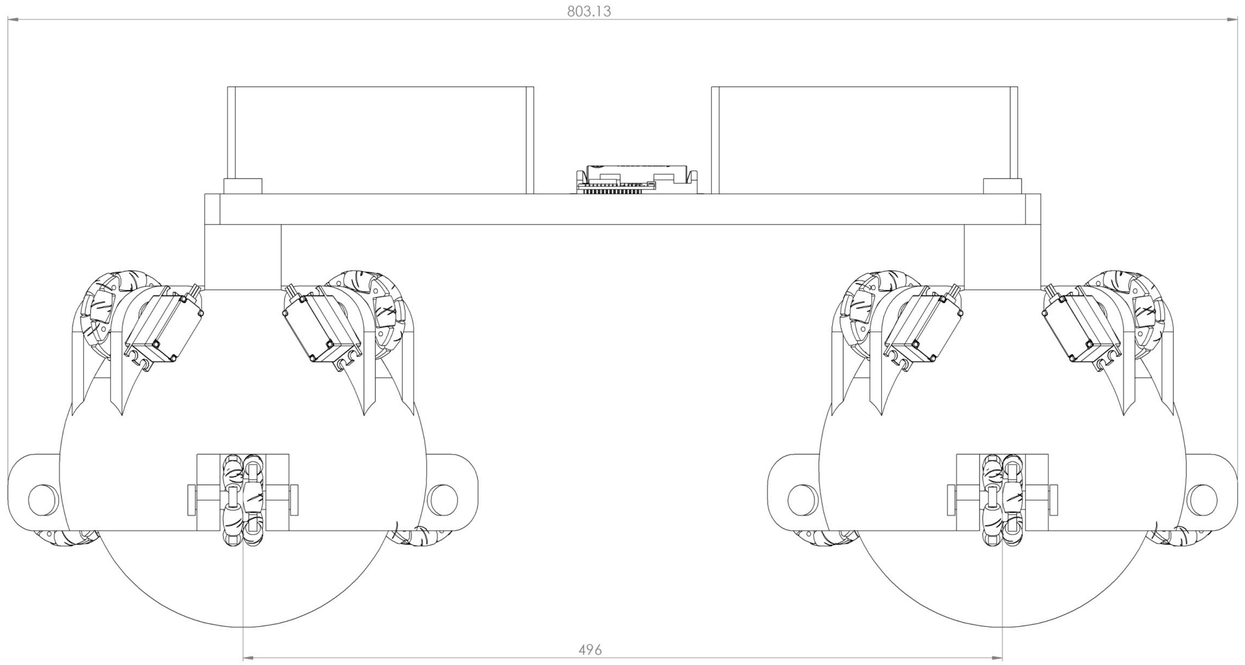


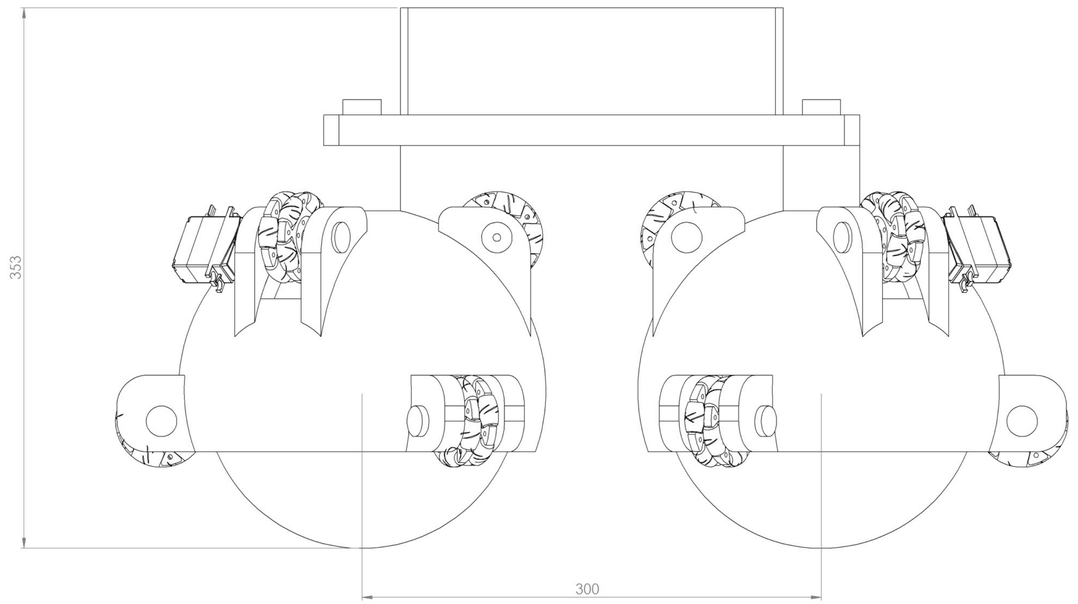


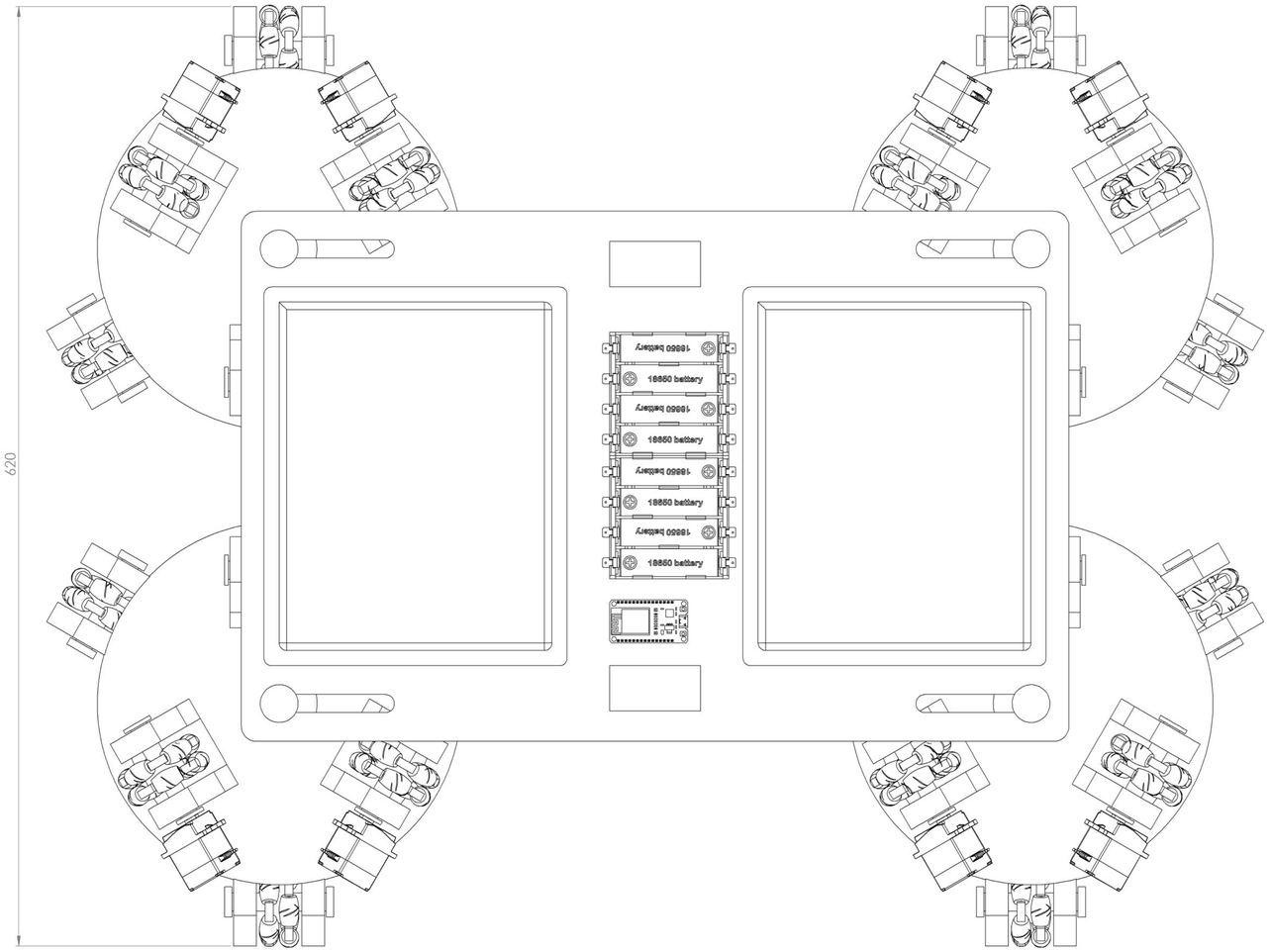


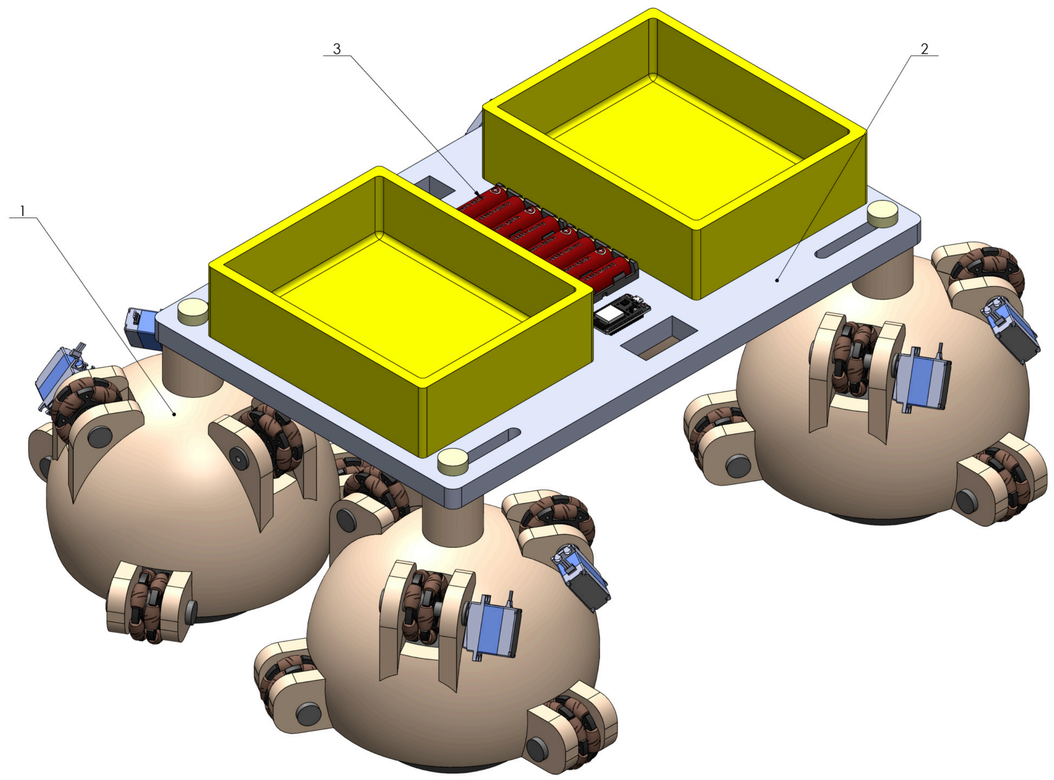
						МР.ПМКм-31.00.00.000	
Зм.	Арк.	№ докум.	Підп.	Дата	Колесо Складальне креслення	Літ.	Маса / Масштаб
Розроб.		Корбеляк Р.В.					
Перев.		Копей В.Б.				Арк.	Аркушів
Т. контр.							
Н. контр.						ІФНТУНГ ПМКм-23-1	
Затв.							

Формат А1









					МР.ПМКм-31.00.00.000	
Зм.	Арк.	№ докум.	Підп.	Дата	Складальне креслення робота	Літ. / Маса / Масштаб
Розроб.		Корбеляк Р.В.				
Перев.		Копей В.Б.				Арк. / Аркушів
Т. контр.						ІФНТУНГ ПМКм-23-1
Н. контр.						
Затв.						

Формат А1

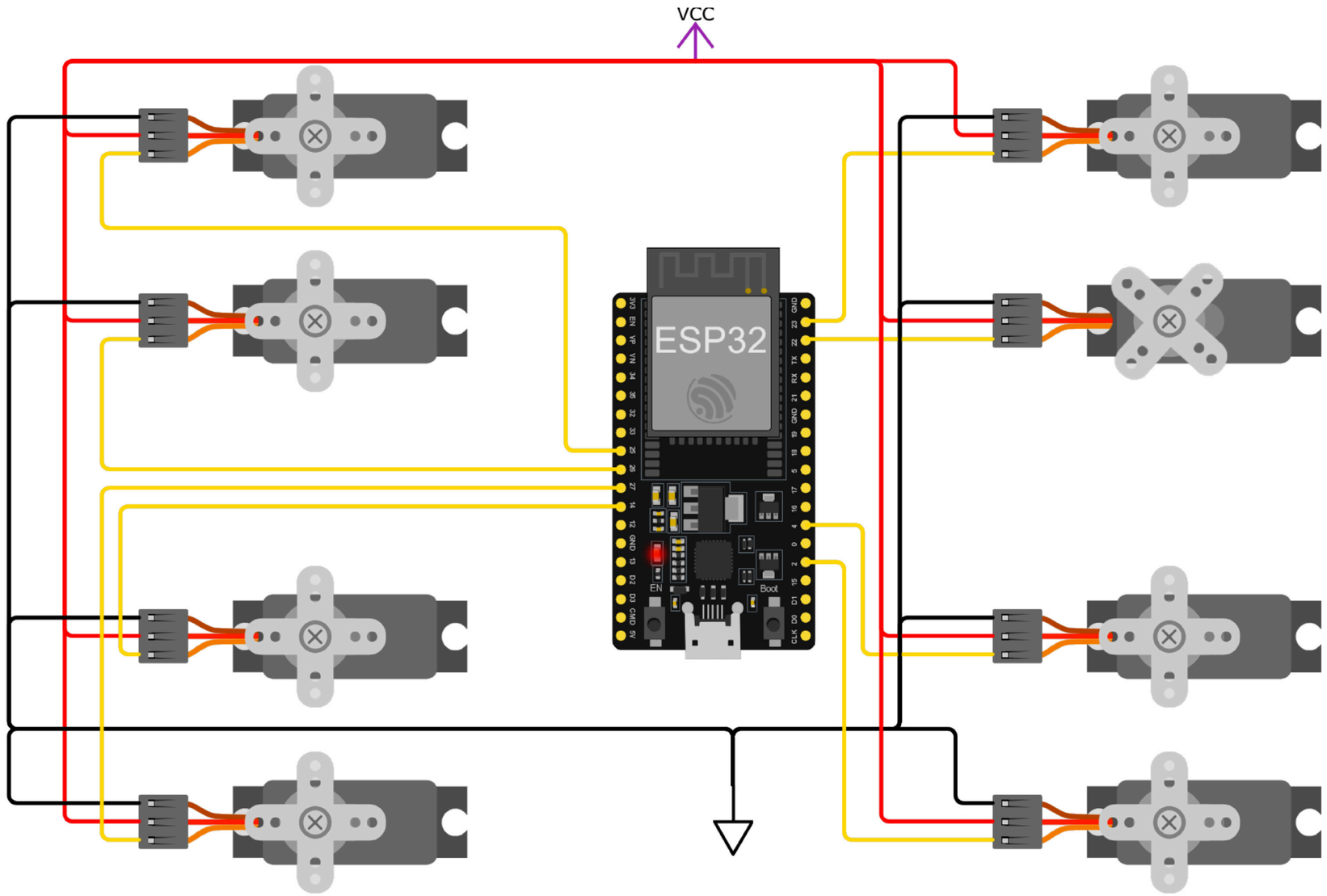


Рисунок 1 - Принципова електрична схема

```
from machine import Pin, PWM
from time import sleep

import socket
s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
s.bind(('192.168.1.116', 8888))
s.listen(1)

sRF1 = PWM(Pin(22), freq=50)
sRF2 = PWM(Pin(23), freq=50)

sRB1 = PWM(Pin(2), freq=50)
sRB2 = PWM(Pin(4), freq=50)

sLF1 = PWM(Pin(26), freq=50)
sLF2 = PWM(Pin(25), freq=50)

sLB1 = PWM(Pin(14), freq=50)
sLB2 = PWM(Pin(27), freq=50)

def run(s):
  s.duty(90) # обертання для серво 360
  sleep(0.1)

def stop(s):
  s.duty(50) # зупинка для серво 360
  sleep(0.1)

# тестування
while True:
  run(sRF1)
  #eval("run(sRF1)")
  sleep(1)
  stop(sRF1)
  sleep(1)

"""
# виконання команд від Wifi-клієнта
while True:
    soc, addr = s.accept()
    print('Server is connected to client', addr)
    command=soc.recv(255) # отримати команду
    print('Client:', command)
    eval(command) # виконати команду
    #eval("run(sRF1))")
    soc.close()
"""
```

						МР.ПМКм-31.00.00.000	
Зм.	Арк.	№ докум.	Підп.	Дата	Принципова електрична схема та керуюча програма	Літ.	Маса	Масштаб
Розроб.		Корбеляк Р.В.						
Перев.		Копей В.Б.				Арк.	Аркушів	
Т. контр.						ІФНТУНГ ПМКм-23-1		
Н. контр.								
Затв.								

Формат А1